

Universidad Carlos III of Madrid

Departamento de Mecánica de Medios Continuos y Teoría de Estructuras



SENIOR THESIS

Industrial Engineering

WEB IMPLEMENTATION OF A COMPOSITE STIFFENED PANEL DESIGN TOOL

(IMPLEMENTACIÓN WEB DE UNA HERRAMIENTA PARA EL DISEÑO DE
PANELES RIGIDIZADOS LAMINADOS DE MATERIALES COMPUESTOS)

Author: Kim Nielsen Martínez

Advisor: Dr. Carlos Navarro Ugena

Departamento de Mecánica de Medios Continuos y Teoría de Estructuras
Universidad Carlos III of Madrid (Spain)

Co-advisor: Dr. Ever J. Barbero

Mechanical and Aerospace Engineering Department
West Virginia University (USA)

Leganés, April 2014

*To mentors and friends that have accompanied me on the journey,
and to those I have yet to meet*

Acknowledgements

I would like to express my gratitude to all these people that helped me all the way through my stage at university, who encouraged me to fulfill my inquisitiveness and pursuit my goals in life.

First of all, I would like to express my gratitude to Professor Ever Barbero for all his help. Thanks for bringing the opportunity to carry out the project in the West Virginia University, for all the concepts I have learnt, and his support and patience, especially in those difficult moments in which I was about to surrender. Apart from the professional relationship, I would also like to thank Professor Barbero for the personal treat I received, for his care and determination in making sure that I did not feel alone or helpless during my stay in the United States. I realize how difficult things could have been without that support.

Second, I would like to thank Professor Carlos Navarro for his assistance in my pursuit of the excellence in education. For the last years I was very committed to try by all means to carry out the project in a foreign country, and I was especially attracted by the US due to the unquestionable prestige of its universities. It would have been much more difficult to accomplish that goal without his aid and support, and I may not have had the chance of meeting incredible people such as Professor Ever Barbero otherwise.

Third, I would like to mention the people who did not provide direct help to the project, but who made all the effort and compromise required along the whole way much easier. Thanks to my parents for all the support and love I received along all these years. So did my sister Karen, one of the persons that best understands the way I am. Though she was not there anymore for supporting me during this last stage of my life, I would also like to thank Marga for making all the five years of university much easier and meaningful. We were a powerful team based on love and confidence, and the mutual support and competition

allowed us to get the best out of us. I doubt I would have become the person I currently am without all of them.

Finally, I would like to mention some incredible people I met during my six-months stay in the small town of Morgantown, in West Virginia. Thanks to Marta, José, and Diego for being my support, they were there when I had to face one of the most difficult situations in my life, and I know it would have been much harder to overcome without them. I would like to mention Eduardo and Ignacio as well, for those trips together and for helping me to fit in such new culture at the beginning. Finally, I would like to thank all the family Barbero for their huge support, love and care. Thanks to Ana, Daniel, and Margaret for being my family there and never let me feel alone.

Abstract

The use of composite materials has grown considerably during recent years, most prominently in the aerospace and automobile industries, as well as in the civil and naval industries. Composite materials display outstanding specific strength and stiffness properties, and a broad capacity for customization. Both qualities have contributed to their application in industries sensitive to the weight of materials such as steel or other alloys. However, composites' behavior is more difficult to predict due to multiple failure modes.

Therefore, complex models and powerful software are normally required to design components made of composite materials, both during the creation of the model and in the results post-processing, adding time to the design process. It would be helpful to automate part of this process in order to speed up the pre-design step. The aim of this project is the development of a tool that will ease the design of composite stiffened panels, reducing the time required to create the model while also allowing fast modifications of a given design in order to maximize the design efficiency. The tool will be able to generate the mesh and geometry from a set of predefined parameters that defines a given panel, and it will compute the critical buckling load of the first five modes, along with the visualization of the deformed panel to ensure quick post-processing. In order to render this tool accessible, it will be implemented in a web environment (CADEC) using advanced features such as object-oriented programming, user authentication and databases.

After implementing such a tool in CADEC, it was used to predict the buckling load of several real cases of stiffened panels. Since the results obtained with CADEC generally coincided with the experimental results analyzed, the tool and its results can be safely validated.

Keywords: Composites, Stiffened Panels, Critical Buckling Load, Stiffened Panels Design, CADEC.

Table of contents

Acknowledgements	i
Abstract.....	iii
1. Introduction.....	1
1.1. Motivation.....	1
1.2. Objectives.....	3
1.3. Project content.....	5
2. Stiffened Panels	7
2.1. Introduction	7
2.2. Stiffened panel configurations.....	8
2.3. Stiffeners	10
3. Buckling.....	12
3.1. Introduction to buckling phenomenon	12
3.2. Motivation in studying buckling.....	14
3.3. Main buckling modes causing stiffened panels failure	15
4. Web programming	17
4.1. Introduction to Object-Oriented Programming, C# and ASP.NET	17
4.2. Object-Oriented Programming (OOP).....	18
4.2.1. History of Object-Oriented Programming.....	18
4.2.2. Object-Oriented Programming concepts	18
4.2.3. Advantages and disadvantages of OOP.....	19
4.3. Web development.....	20
4.3.1. History of web development.....	20
4.3.2. The “.NET” framework	21
4.4. Programming language selection.....	22
5. The Design tool	23
5.1. Coordinate Systems (CSYS) utilized.....	24
5.1.1. Default CSYS of the mesh generator	25
5.1.2. Calculation of local CSYS for each element by the solver	25
5.1.3. Final CSYS used by CADEC	27

5.2.	The Mesh Generation tool	29
5.2.1.	Mesher of flat surfaces: FlatRecGeo.cs class	29
	A. Aim and arguments	29
	B. Methods	32
5.2.2.	Mesher of concave surfaces: CurvedRecGeo.cs class	36
	A. Aim and arguments	36
	B. Methods	37
5.2.3.	Seeding of panel mesh: MeshParameters.cs class.....	38
	A. Aim and arguments	38
	B. Methods	38
5.3.	The Problem Builder tool	39
5.3.1.	Panel mesh generator: SPmshBuilder.cs class	39
	A. Aim and arguments	39
	B. Methods	40
5.4.	The Load and Boundary Conditions manager tool.....	44
5.4.1.	Handling loads and boundary conditions: INPUTdata.cs class.....	45
	A. Aim and arguments	45
	B. Methods	46
5.5.	The solver: BMI3.....	51
5.6.	The visualization tool	53
5.6.1.	Results post-processing using gmsh: POSdata.cs class	53
	A. Aim and arguments	53
	B. Methods	54
6.	Web Environment	55
6.1.	CADEC.....	55
6.2.	Databases and stored procedures	57
7.	Implementation	58
7.1.	User's view	58
7.2.	Programmer's view	60
7.2.1.	Kernel Project: definition of CADEC's objects	60
7.2.2.	KernelFortran Project: CADEC's solver.....	67
7.2.3.	Web Project: CADEC's pages definition	67
	A. My Documents	68
	B. Chapters	76

7.2.4. WebReusables Project: code reuse across pages	84
8. Visualization	89
8.1. The need of a visualization tool	89
8.2. WebGL technology	90
8.2.1. Renderer pipeline of WebGL	91
8.2.2. Definition of the different variables used by WebGL	94
8.2.3. Description of the JavaScript code	96
9. Tool validation	98
9.1. Validation with theoretical expressions of Classical Laminated Plate Theory (CLPT) ..	99
9.1.1. Flat Plates subject to compressive loads	99
A. Flat plate with simply supported boundary	100
B. Flat plate with clamped boundary	105
C. Resolution of previous cases applying symmetric BC	107
D. Flat plate subject to biaxial loading	110
9.2. Validation of a curved panel subject to compression analyzed with ANSYS	112
9.2.1. Curved panel analyzed with ANSYS	112
9.2.2. Curved panel analyzed with CADEC	113
9.3. Validation of flat stiffened panels using experimental data	117
9.3.1. Flat stiffened panel with hat stringers	117
9.3.2. Flat stiffened panel with JC stringers	121
9.3.3. Flat stiffened panels with J stringers	125
A. Experimental results	125
B. Results of CADEC simulation	127
9.4. Validation of curved stiffened panel analyzed with Abaqus	134
9.4.1. Curved T-stiffened panel analyzed with Abaqus	134
9.4.1. Curved T-stiffened panel analyzed with CADEC	136
10. Conclusions and future work	138
10.1. Conclusions	138
10.2. Possible future work	140
Bibliography	141
Appendices	146
Appendix A: MSH file	146
Appendix B: BMI3 INP file	148
Appendix C: BMI3 DAT file	150

Appendix D: Abaqus INP file..... 152

Appendix E: Stored procedures and SQL language 156

List of figures

Chapter 2

Figure 2.1: Example of unidirectional stiffened panel [54].	8
Figure 2.2: Various configurations of stiffened panels showing the primary and secondary stiffeners [54].	9
Figure 2.3: Example of grid-stiffened panels [28].	9
Figure 2.4: Main parts of a stiffener profile.	10
Figure 2.5: Different types of profiles accepted by CADEC.	11

Chapter 3

Figure 3.1: Primary and secondary paths representation. Source: Image 4.1 of [5].	13
Figure 3.2: Example of local buckling -skin buckling-. Source: Figure 2 of [26].	15
Figure 3.3: Example of global buckling -skin and stringers buckling-. Source: Figure 2 of [26]. .	15
Figure 3.4: Example of tripping -stiffener torsional buckling-. Source: Figure 2 of [26].	16

Chapter 5

Figure 5.1: Default Coordinate System employed by our tools, different from the one deployed within CADEC environment. Created using Solid Edge®.	25
Figure 5.2: Coordinate System inconsistency when using default CSYS.	27
Figure 5.3: Final CSYS used in CADEC. Created using Solid Edge®.	28
Figure 5.4: Local elements CSYS in CADEC. Created using Solid Edge®.	28
Figure 5.5: Flat parallelogram meshed with second order quadrangular elements using CADEC. Image visualized using gmsh [21].	30
Figure 5.6: Flat parallelogram meshed with first order quadrangular elements using CADEC. Image visualized using gmsh [21].	31
Figure 5.7: Flat parallelogram meshed with first order triangular elements using CADEC. Image visualized using gmsh [21].	31
Figure 5.8: Meshing sequence for a flat parallelogram -nodes generation-.	33
Figure 5.9: Joint representation of two subparts sharing a common edge.	34
Figure 5.10: Meshing sequence for a flat parallelogram -first order quadrangular elements generation-.	35

<i>Figure 5.11: Meshing sequence for a flat parallelogram –second order quadrangular elements generation-.....</i>	<i>35</i>
<i>Figure 5.12: Meshing sequence for a flat parallelogram –first order triangular elements generation-.....</i>	<i>36</i>
<i>Figure 5.13: Curved parallelogram meshed with our application. Image visualized using gmsh [21].</i>	<i>37</i>
<i>Figure 5.14: Definition of some basic parameters required to create a stiffened panel. Created using Solid Edge®.....</i>	<i>40</i>
<i>Figure 5.15: Schematic representation of the sequence followed to create the geometry and mesh of the main seven profiles of the application.</i>	<i>43</i>
<i>Figure 5.16: Definition of the rotational degrees of freedom used by the tool. Source: Figure 5.2 of [6].</i>	<i>45</i>
<i>Figure 5.17: Schematic representation of additional constraints required to ensure problem solution of panels subject to in-plane compression.</i>	<i>48</i>
<i>Figure 5.18: Schematic representation of additional constraints required to ensure problem solution of panels subject to in-plane shear loading.....</i>	<i>48</i>
<i>Figure 5.19: Load share for each type of element.....</i>	<i>50</i>
<i>Figure 5.20: CADEC results for 9-node fully integrated/relaxed element, with no zero-energy mode.....</i>	<i>52</i>

Chapter 7

<i>Figure 7.1: Main page of the web application CADEC.....</i>	<i>58</i>
<i>Figure 7.2: Scheme of the different projects forming CADEC.....</i>	<i>60</i>
<i>Figure 7.3: Files added to Kernel Project for the web implementation of our application in CADEC.....</i>	<i>61</i>
<i>Figure 7.4: Files required for the pages in My Documents –CADEC-.....</i>	<i>68</i>
<i>Figure 7.5: My Documents/Stiffened Panels/Default page of CADEC.....</i>	<i>68</i>
<i>Figure 7.6: My Documents/Stiffened Panels/Edit page of CADEC.</i>	<i>70</i>
<i>Figure 7.7: Layout of flat and curved stiffened panels containing the main dimensions. Created using Solid Edge®.....</i>	<i>71</i>
<i>Figure 7.8: Flat and curved stiffened panel images. Created using Solid Edge®.....</i>	<i>71</i>
<i>Figure 7.9: Stringer properties definition in CADEC, showing the “Omega” profile example.</i>	<i>72</i>
<i>Figure 7.10: Different offset configurations. Left figure shows a stiffened panel with offset zero whereas right figure shows a positive offset. Created using CADEC output.</i>	<i>72</i>
<i>Figure 7.11: My Documents/Stiffened Panels/Visualization page of CADEC.....</i>	<i>75</i>
<i>Figure 7.12: Files required for the pages in Chapters -CADEC-.</i>	<i>76</i>

<i>Figure 7.13: Chapters/Stiffened Panels/Buckling page of CADEC.</i>	78
<i>Figure 7.14: Loading bar appears while the problem is being solved on the server.</i>	79
<i>Figure 7.15: Chapters/Stiffened Panels/Visualization page of CADEC.</i>	82
<i>Figure 7.16: Files enabling code-reuse among the different pages related to Stiffened Panels in CADEC.</i>	85
<i>Figure 7.17: Part of a page in CADEC generated by a single file, illustrating the power of code reuse.</i>	86
<i>Figure 7.18: Files that manage the integration of images in a table row, used in the different pages related to stiffened panels in CADEC.</i>	87

Chapter 8

<i>Figure 8.1: Two schemes representing the renderer pipeline of WebGL. Sources: [1] and [22].</i>	93
<i>Figure 8.2: Scheme of a nine-node mesh element division into triangles for visualization.</i>	95
<i>Figure 8.3: Texture coordinates for each node in a nine-node element.</i>	96

Chapter 9

<i>Figure 9.1: Buckling results for a flat plate with simply supported boundary subject to uniaxial compression using CADEC.</i>	102
<i>Figure 9.2: Buckling results for a slender flat plate with simply supported boundary subject to uniaxial compression using CADEC.</i>	104
<i>Figure 9.3: Buckling results for a flat plate with clamped boundary subject to uniaxial compression using CADEC.</i>	106
<i>Figure 9.4: Deformed shape of a flat plate with simply supported boundary applying symmetric boundary conditions using CADEC.</i>	108
<i>Figure 9.5: Deformed shape of a flat plate with clamped boundary applying symmetric boundary conditions using CADEC.</i>	109
<i>Figure 9.6: Deformed shape of a flat plate with simply supported boundary under biaxial compression using CADEC.</i>	111
<i>Figure 9.7: Layout of the curved panel analyzed with ANSYS®. Source: Figure 3 of [30].</i>	112
<i>Figure 9.8: Buckling results using CADEC of the curved panel subject to axial compression described in [31], using symmetry.</i>	115
<i>Figure 9.9: Deformed shape of the curved panel subject to axial compression described in [31] (whole model) using CADEC.</i>	116
<i>Figure 9.10: Layout of the stiffened panel with three hat stringers. Source: Figure 2 of [17].</i>	117
<i>Figure 9.11: Hat stringers profile layout. Source: Figure 1 of [18].</i>	117
<i>Figure 9.12: Buckling results of the stiffened panel with three hat stringers using CADEC.</i>	120

<i>Figure 9.13: Layout of the stiffened panel with three JC stringers. Source: Figure 2 of [18]. ...</i>	<i>121</i>
<i>Figure 9.14: JC stringers profile layout. Source: Figure 1 of [18]......</i>	<i>121</i>
<i>Figure 9.15: Buckling results of the stiffened panel with three JC stringers using CADEC.</i>	<i>124</i>
<i>Figure 9.16: Layout of the three different configurations of stiffened panel using J stringers. Source: Figure 2 in [11]......</i>	<i>125</i>
<i>Figure 9.17: Buckling results of case 1: stiffened panel with three J stringers using CADEC. ...</i>	<i>130</i>
<i>Figure 9.18: Buckling results of case 2: asymmetric stiffened panel with four J stringers using CADEC.....</i>	<i>131</i>
<i>Figure 9.19: Buckling results of case 3: symmetric stiffened panel with three J stringers (half model) using CADEC.</i>	<i>132</i>
<i>Figure 9.20: Buckling critical load for ideal and imperfect paths. Source: Edition of Image 4.1 in [5].</i>	<i>133</i>
<i>Figure 9.21: Buckling results of a curved stiffened panel with four T stringers using Abaqus®.</i>	<i>135</i>
<i>Figure 9.22: Buckling results of a curved stiffened panel with four T stringers using CADEC. ..</i>	<i>137</i>

List of tables

Chapter 5

Table 5.1: Degrees of freedom restrained for each boundary condition type depending on the side. 47

Table 5.2: Additional degrees of freedom restrained depending on the type of applied load to ensure that the panel has no rigid body motion. 47

Chapter 9

Table 9.1: Lamina properties for all flat plates cases..... 99

Table 9.2: Laminate properties used in all flat plates cases..... 100

Table 9.3: Summary of the resulting coefficients of the bending stiffness matrix $[D]$, which are necessary to compute the theoretical critical buckling load. 100

Table 9.4: Loads, BC and geometry input for plate A in CADEC. 101

Table 9.5: Loads, BC and geometry input for plate B in CADEC. 105

Table 9.6: Loads, BC and geometry input for plate D.1 in CADEC. 107

Table 9.7: Loads, BC and geometry input for plate D.2 in CADEC. 108

Table 9.8: Loads, BC and geometry input for plate E in CADEC. 110

Table 9.9: Mechanical properties of the lamina used in CADEC for the curved panel case. 113

Table 9.10: Loads, BC and geometry input for the curved plate in CADEC. 114

Table 9.11: Geometry of the Hat-stiffened panel introduced in CADEC..... 118

Table 9.12: Mechanical properties of the lamina for the Hat-stiffened panel..... 118

Table 9.13: Stacking sequence (SS) of the different laminates forming the Hat-stiffened panel, and their area of application..... 118

Table 9.14: Load and BC applied on each side of the Hat-stiffened panel..... 119

Table 9.15: Geometry of the JC-stiffened panel introduced in CADEC. 122

Table 9.16: Mechanical properties of the lamina for the JC-stiffened panel. 122

Table 9.17: Stacking sequence (SS) of the different laminates forming the JC-stiffened panel, and their area of application..... 122

Table 9.18: Load and BC applied on each side of the JC-stiffened panel. 123

Table 9.19: Geometry of the three experimental different J-stiffened panels. 126

Table 9.20: Mechanical properties of Aluminum used in the three J-stiffened panels. 126

Table 9.21: Experimental values for the critical buckling load obtained by [11]. 127

Table 9.22: Geometry of the three JC-stiffened panels introduced in CADEC. 127

<i>Table 9.23: Mechanical properties of the lamina for the JC-stiffened panels modeled with CADEC.....</i>	<i>128</i>
<i>Table 9.24: Load and BC applied on each side of the three different J-stiffened panels.....</i>	<i>128</i>
<i>Table 9.25: Comparisson between the experimental buckling loads [11] and the results provided by CADEC for the three different panels.....</i>	<i>129</i>
<i>Table 9.26: Geometry of the curved T-stiffened panel defined in Abaqus®.....</i>	<i>134</i>
<i>Table 9.27: Geometry of the curved T-stiffened panel introduced in CADEC.....</i>	<i>136</i>
<i>Table 9.28: Load and BC applied on each side of the curved T-stiffened panel.....</i>	<i>136</i>

Chapter 1

1. Introduction

1.1. Motivation

The use of composite materials has grown considerably during recent years, especially in the aerospace and automobile industries, as well as in the civil and naval industries [38, 54]. One of the most general applications of composites is in the form of plates or panels, such as those used in the body of a car or the fuselage and wings of an airplane, due to composites' versatility and relative ease of manufacturing. Most of these panels include some stiffeners in order to increase out-of-plane structural stiffness [42, 49].

Composite materials are valued for their high specific strength or specific stiffness, their corrosion, magnetic and conductive resistances, their fatigue life, and the possibility of customization of the material. This last advantage refers to the wide range of parameters that can be modified during the design step in order to obtain a material with the precise mechanical properties desired. The properties of conventional materials such as steel or aluminum are quite standard. In order to increase stiffness in a structure using these materials, engineers typically modify non-material-dependent parameters such as geometry, layout and quantity of material employed. In contrast, composites provide a much higher level of customization: reinforcement material selection, matrix material selection, architecture (fiber

length and orientation, particles, fabrics, volume of reinforcement, stacking sequence, etc.) [41]. With all these variables available to designers, the optimization and efficiency of the design can reach levels impossible with conventional materials. Within the aerospace and automobile industries, there is a strong additional incentive to use lighter components and realize important savings in fuel consumption and emissions [44].

However, the analysis of composites' behavior is complicated by their nonlinearity, anisotropy, and multiple modes of failure [41]. This difficulty usually requires specific software in order to carry out Finite Element Analysis (FEA) of the components, since there are few cases that could be solved manually [31]. The creation of such Finite Element (FE) models and the verification of the results require significant time due to their complexity. Indeed, in the design step, during which several important decisions concerning sizing or material selections are made, this increased time is very apparent.

Though there already is a complete set of powerful software capable of running FE analysis on these kinds of structures, the process normally takes too long. Under these circumstances, it is necessary to expedite the stiffened panels design process through a new application [43].

The first attempt to provide such an application was the desktop application SPD (*Stiffened Panel Design*) [53]. It is capable of calculating the critical buckling load of flat and curved plates as well as stiffened panels under the following specifications:

- It is capable of performing buckling analyses on plates and stiffened panels, whether flat or slightly curved.
- There are six different profiles available as stringers.
- Normal or shear loading configurations as well as four types of boundary conditions (simply supported, clamped, free or symmetry) can be applied to each edge of the panel. This allows modeling of most real life configurations.
- Composites and isotropic materials can be used to model the panels.

SPD accomplishes its function quickly and with outstanding accuracy. However, a few persistent weaknesses indicated space for a more complete, efficient and portable application. The disadvantages of SPD include:

- As a Windows application, SPD has the advantage of offline use, but is operating system sensitive.
- SPD relies upon the freeware gmsh [21] to generate the geometry and mesh of the panel's Finite Element model, and to visualize the first buckling mode. This software is very powerful and achieves high-quality meshes, but requires the user to install it in the computer in order to use SPD. In addition, scripts of code in the form of text files are required to create the geometry of each type of profile. If these files are altered or eliminated by the user, the application's integrity is compromised.

- SPD uses an executable version of BMI3 [6] to solve the Finite Element problem. Since communication between the SPD executable, BMI3, and gmsh is accomplished using files, it becomes tedious to store several analyses for future use, and the user is required to manage the files directly, outside the application environment.
- Material properties are introduced by storing the ABDH matrices of each of the four possible materials in text files, an error-prone and tedious process. Additionally, SPD does not handle computation of such parameters.

These issues prompted development of a new tool based on SPD to perform similar computations without the disadvantages.

1.2. Objectives

The aim of this project is to create an application that will assist preliminary design of composite stiffened panels, and address the needs outlined in the previous section. To accomplish this, the problem is simplified to a number of parameters describing geometry, materials, loads and boundary conditions. The application will solve the buckling problem and provide as a result the critical buckling load of the first five modes, as well as the deformed shape of the panel for the first mode. This application will be implemented in a web environment usable from almost any device, and independent of external software such as gmsh [21].

In order to achieve this main goal, some partial objectives are set:

- A mesh generator is required. Though some extant freeware as gmsh [21] can perform this operation offline at a user's computer, this program will be created from scratch, for embedding in a web environment. Unlike SPD, this program will control the mesh while independent of external software. Furthermore, the main application and the mesh generator will communicate programmatically, eliminating the need for files and accelerating the process. This tool should be able to generate the mesh for a generic stiffened panel based on some geometric parameters. This tool will automate the creation of the FE model to reduce time required to create such a panel to the minimum necessary for defining some main parameters. The tool should be able to create flat and curved panels, with or without stiffeners. There will be at least six different profiles for the stringers.

- Another tool is required to manage all problem input data, including the mesh generator tool management, and prepare it to be sent to the solver. Some of the main tasks include materials properties and assignment to the corresponding elements, translation of the BC and Loads per side to the actual values applied on the nodes in the mesh, and management of the results provided by the solver. This code will be placed in a new class so that it can be used from a desktop application or a web environment.
- This tool will be implemented within an existing site: CADEC (Computer Aided Design Environment for Composites) [2]. This online composite materials analysis application has been developed by Professor E. J. Barbero and is hosted by the Department of Mechanical and Aerospace Engineering of West Virginia University. The project objective is therefore to add a new module in charge of creating stiffened panels and solving the corresponding buckling problem to this web application. CADEC provides three main advantages over a desktop application such as SPD. First, as a web environment, it is portable over different operating systems and devices, including smartphones and tablets. Second, the infrastructure of CADEC includes user authentication and relational database use, so that all the panels created can be safely stored for later use. Third, there are extant modules in CADEC in charge of defining fibers, matrices, laminas or laminates, with which the new module would interact in order to obtain the material properties required for the panel definition. This will simplify introduction of material properties for analyzing the panels and results in a more robust application.
- A visualization tool will be added to show the deformed shape of the panel. This could be done by extant free software such as gmsh [21], which displays results at the user's computer offline through SPD, or by developing a specific tool in order to visualize results online. The second option has been chosen as it provides accessibility and faster results. By seeing a graphic representation of the deformed shape of the buckling modes, the designer can identify the weakest areas of the panel and better mitigate the problem.
- The tool will be validated using classic plate theory of several simple examples and some experimental data of stiffened panels. Since the solver used by SPD and CADEC will be virtually the same, accuracy of results should be similar to that of SPD.

1.3. Project content

This project is divided into 10 chapters, this introduction being the first. Brief descriptions of each following chapter follow:

The second chapter contains a summary of the literature related to stiffened panels. The characteristics of these structural elements are defined, as well as their different constituent parts. Moreover, a description of the stiffening elements is presented.

The third chapter presents a general description of buckling phenomenon (structural instability) in structural elements. Then, its relevance in stiffened panel design is highlighted. Finally, the different types of instability affecting stiffened panels are reviewed and explained.

The fourth chapter introduces the relevant web programming technologies, specifically covering object-oriented programming and web development using ASP.NET applications. Reasons for the use of these particular programming languages and technologies are included.

The fifth chapter provides a detailed explanation of the design tool implemented in this project, describing its different components and their functioning. This chapter focuses on the different tools used to solve the design problem: mesh generation, load and BC application, or solver.

The sixth chapter describes the web environment in which the application will be implemented, which is CADEC [2]. It also introduces some relevant features of data storage in a database, which are required to explain the actual implementation of the application.

The seventh chapter describes the process of implementing the application in the web environment (CADEC). First, it analyzes how the web application will be structured for the final user. Then, it focuses on the programming challenges of the implementation, and its relationship to the structure of CADEC. CADEC is divided into different projects, each of which accomplishes a different mission. The content of each project is reviewed in this chapter.

The eighth chapter focuses on WebGL [22], the technology deployed for visualizing results. After introducing the technology, a further explanation follows to illustrate how the image is rendered in the computer. Finally, the chapter describes the different variables involved in this process as well as a brief review of the JavaScript code defining the tool.

The ninth chapter verifies that the results provided by CADEC are valid and accurate. In order to do so, plates with different boundary condition and load configurations are first analyzed and compared with theoretical expressions. Then, one case of a curved plate is compared with another finite element model in the literature. Finally, stiffened panels with different geometry and stringer profiles are analyzed using CADEC and the results are compared with experimental data available in the literature.

The tenth chapter states the main conclusions drawn from the creation and validation of the application. It also suggests possible lines of future work.

Finally, the bibliography used for the development of the project is presented. Furthermore, the document includes some appendices providing further information on the structure of the different files (MSH, INP, DAT) that the application can generate, a deeper explanation of SQL language, and the structure of the stored procedures defined in this project to communicate with the database.

Chapter 2

2. Stiffened Panels

2.1. Introduction

Designers in the aerospace, automobile, naval and civil engineering fields face a strong motivation to reduce the amount of material used in order to lighten the total weight of a structure. In aerospace and automobile engineering, this motivation is mainly driven by the reduction in vehicular fuel consumption, or efficiency improvement.

Plates are one way of achieving this goal. Stiffened panels, and plates in general, belong to the wider group of slender or thin-walled components. These components can be manufactured with metals or composites. Metals are normally easier to analyze due to isotropy and homogeneity, but their weight limits their applicability. On the other hand, composites pose additional difficulties to analysis due to anisotropy, material joints, multiple failure modes, etc. However, they normally allow a higher level of customization and efficiency, resulting in the best specific properties.

One of the problems with this type of structure is slenderness. Despite non-stiffened panels' high capacity to withstand in-plane loading configurations, their capacity for out-of-plane loading, such as bending, is very poor. The only way to increase their stability is by

increasing the plate thickness. For this reason plates (which in a stiffened panel are referred as *skin*) are normally accompanied by some beams (known as *stiffeners* or *stringers*) in one or several directions, so that inertia is increased without an important penalty in total weight, and the thickness of the plate can be kept as thin as possible).

2.2. Stiffened panel configurations

The panels may be initially flat or curved. Unidirectional stiffened panels have several applications in ship and airplane construction. In the latter, the cover of the fuselage and the wings are manufactured with this structural element [13, 38]. However, cross-stiffened panels represent the most common structural units used in ship construction and civil engineering [54]. These stiffened panels are composed of plate panels with primary and secondary stiffening structural members. The secondary stiffeners are supported by primary structural members spaced at ample distances from each other in order to ensure that the plate panels will have adequate strength and stiffness when loaded. The main function of the stringers (longitudinal stiffeners) is to withstand important axial loads (tension and compression), whereas the main goal of transversal structural profiles (ribs) is to add stiffness and in-plane resistance. For that reason, ribs are not usually modeled in the analysis since they are considered very rigid structural elements (bulkheads) delimiting the panel.

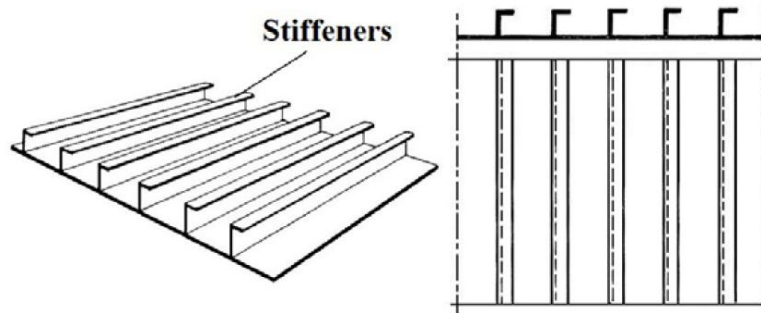


Figure 2.1: Example of unidirectional stiffened panel [54].

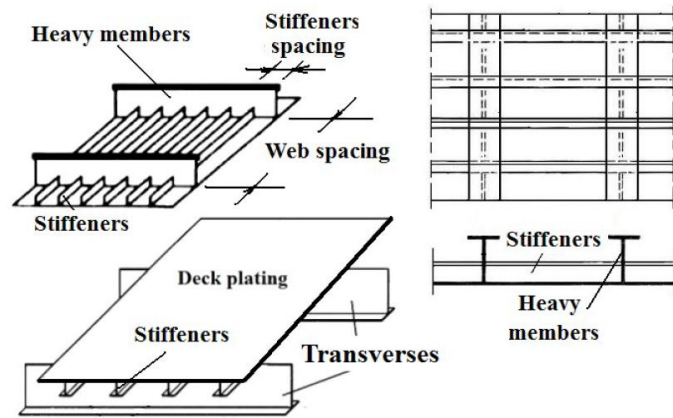


Figure 2.2: Various configurations of stiffened panels showing the primary and secondary stiffeners [54].

Apart from unidirectional and cross-stiffened panels, which set stiffeners in longitudinal and transversal directions, there are also grid-stiffened panels in which stiffeners can take more directions. However, their use is not as widespread as that of previously described types.

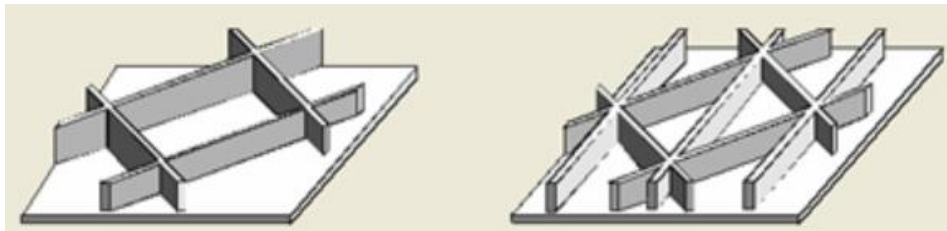


Figure 2.3: Example of grid-stiffened panels [28].

According to Niu [43], stiffened panels can be classified in two main types regarding the joint between stringers and skin: skin-stringer panels and integrally-stiffened panels. In the first case, skin and stringers are joined by lines of bolts, welds or similar joints [44], or by pasting or co-curing them [52]. In the second case, skin and stringers are manufactured at the same type, becoming a unique structure. The difference between types will be further explained in the next section.

2.3. Stiffeners

There are multiple profiles of stringer types. Similar to beams, these profiles are characterized by a high inertia moment with small area in order to reduce the amount of material employed. In order to achieve this, they use bottom flanges which are joined to the skin, one or several webs, and in some cases top flanges. The main function of the webs is to absorb shear forces, and for that reason they should be as perpendicular to the plate as possible. On the other hand, flanges are mainly subject to normal stresses due to bending moments [40, 41]. By the use of a web and flanges the inertia of the plate increases considerably, improving the stiffness under out-of-plane loading conditions [7, 13].

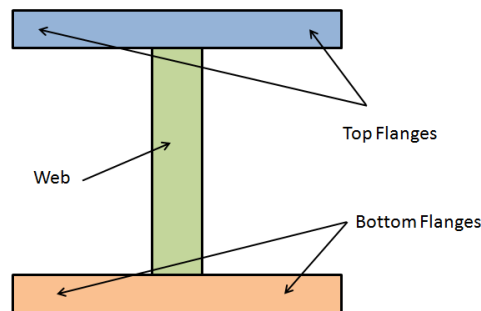


Figure 2.4: Main parts of a stiffener profile.

Sometimes the stringers and the skin are manufactured at the same time, forming a unique component with common materials and properties. Contrary to *skin-stringer panels*, these are called *integrally-stiffened panels*. The advantages of this type are weight reduction of the panel, as the bottom flange is actually the skin of the panel, and increased buckling resistance and homogeneity, since stringer-skin joints are suppressed [44].

Profiles can be opened or closed. When compared to closed profiles, opened profiles' main disadvantage is their low torsional stiffness, which increases the chances of tripping (explained further in this chapter) [16]. However they are usually lighter in weight, and so should not be easily rejected.

Despite the wide variety of possible profiles in the literature, there are six that account for most cases: "T", "L", "Z", "C", "I" and "hat" ("Omega"). According to the literature the most commonplace are "I" (also known as "double-T") and "hat". The use of a specific profile depends on the case under consideration, since all of them present unique advantages and

disadvantages. In order to model one of the cases in the literature to validate the tool, an additional “JC” type was added to the application.

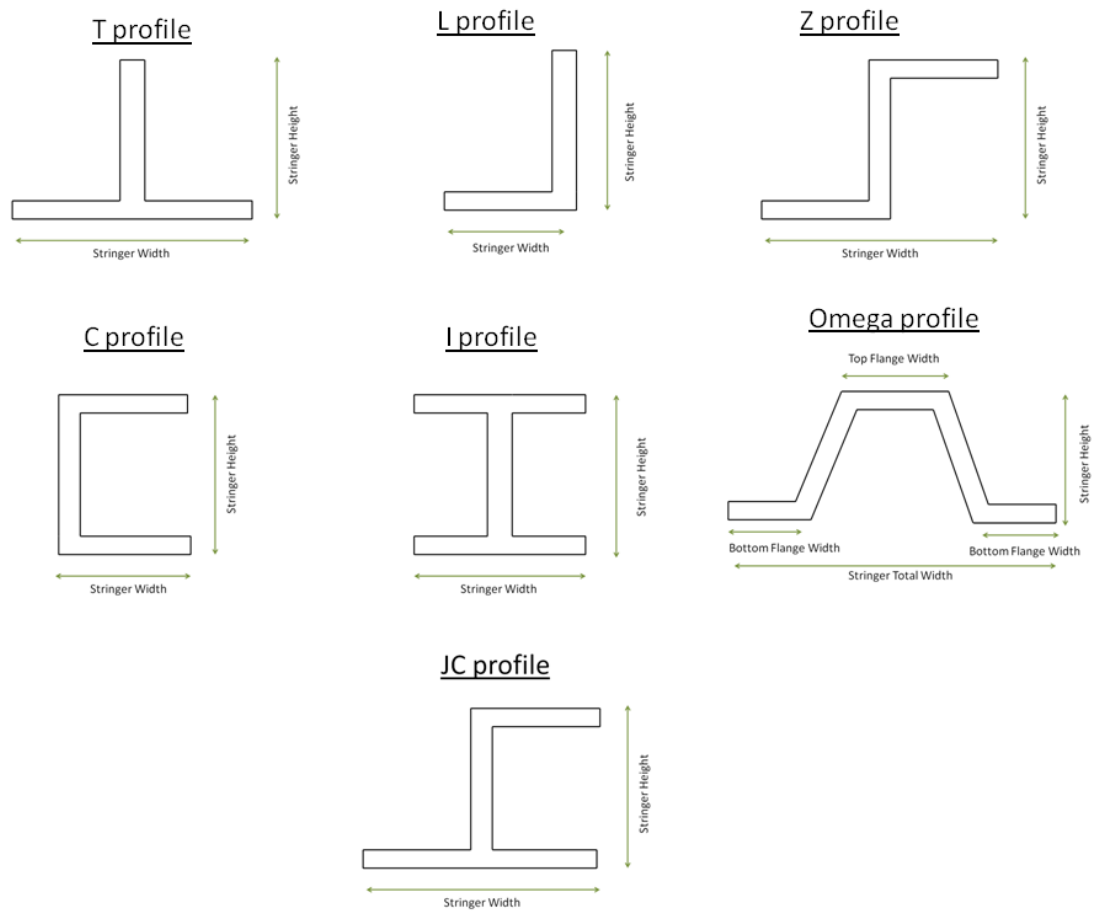


Figure 2.5: Different types of profiles accepted by CADEC.

Chapter 3

3. Buckling

3.1. Introduction to buckling phenomenon

There are two major problems leading to the sudden failure of a mechanical component: material failure and structural instability (so-called buckling) [56]. Material failures are normally determined by the yield stress for ductile materials, or the ultimate stress for brittle materials. They are material properties, and therefore geometry is not specifically addressed when defining them. Predicting material failure may be accomplished then using finite element analysis, which is mainly solving a linear algebraic system for the unknown displacements: $\mathbf{K} \cdot \boldsymbol{\delta} = \mathbf{F}$. The strains and corresponding stresses derived from the analysis are then compared to design stress (or strain) allowable everywhere within the component. If there are regions where these limits are exceeded, it is assumed that material failure has occurred [4, 5].

On the other hand, the load at which buckling occurs depends on the stiffness of a component, not upon the strength of its materials. Buckling refers to the loss of stability of a component due more to geometrical effects rather than material failure [55]. This loss of stability usually occurs within the elastic range of the material. These phenomena are governed by different differential equations [8]. Instead of relying on the usual finite element

analysis, buckling is modeled by a finite element eigenvalue-eigenvector solution, $[\mathbf{K} + \lambda_m \mathbf{K}_F] \cdot \boldsymbol{\delta}_m = \mathbf{0}$, where λ_m is the buckling load factor for the m-th mode, \mathbf{K}_F is the additional geometric stiffness due to stresses caused by the loading \mathbf{F} , and $\boldsymbol{\delta}_m$ is the associated buckling displacement shape for the m-th mode. The spatial distribution of the load is important, but its relative magnitude is not. The buckling calculation gives a multiplier that scales the magnitude of the load (upwards or downwards) to the value required to cause buckling. In other words, once the solution has been found it is possible to obtain the m-th buckling critical load by multiplying the m-th eigenvalue times the applied load F .

In order to better understand this phenomenon, let us consider one of the most well-known examples of buckling: “Euler buckling” [24, 56]. It takes place when compressive axial loads are applied to a slim beam. During the first stage of the loading process, the bar squeezes axially, reducing its total length and increasing the cross section due to Poisson effect. The deformation of the structure before buckling occurs is called primary path. The slightest imperfection will make the column buckle suddenly at a specific load (known as critical load), experiencing some bending (lateral displacement). What happens after the column reaches its critical load depends enormously on the support conditions. The behavior of the structure after buckling has occurred is called post-buckling. Some types of structures do not experience deformation in the shape of the buckling mode before buckling actually takes place. Euler buckling is an example. In this case, buckling occurs at a bifurcation point, which is the intersection of the primary path with the post-buckling path [5].

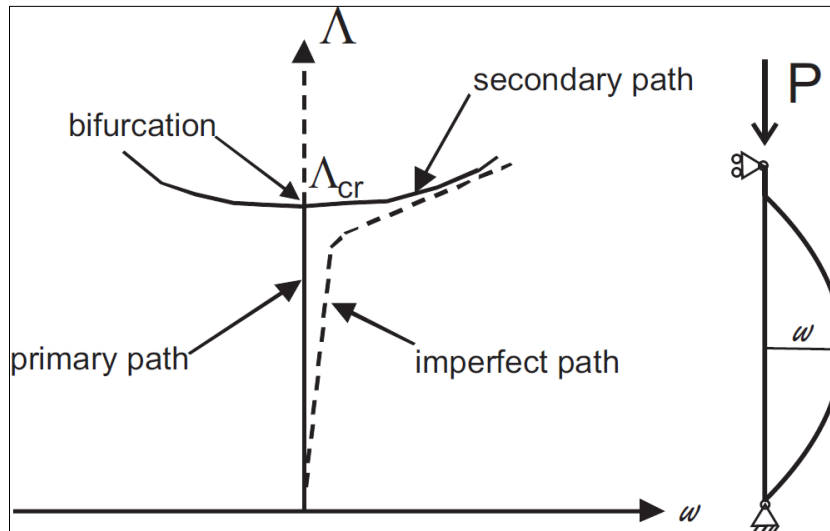


Figure 3.1: Primary and secondary paths representation. Source: Image 4.1 of [5].

To illustrate how geometry influences buckling instability, the critical load of the column increases inversely proportional to the square of its length. Boundary conditions also play an important role in buckling. In Euler’s example, buckling motion will vary by a factor of four depending on how both ends are restrained.

The column's secondary path is quite stable, represented by a nearly horizontal curve, indicating that it can take almost no higher load after buckling. A slight increase in load will laterally deform the column so much that the material will fail and the system will collapse. However, plates which are simply supported experience significant stiffening on the secondary path, indicating that the secondary path still has some slope [9].

3.2. Motivation in studying buckling

The application developed in this project will solve the buckling problem instead of running the usual FEA used to check material failure. Stiffened panels, and plates in general, belong to the wider group of slender or thin-walled components. In fact, most composite structures are thin-walled, for three main reasons [5]:

- Composites are stronger than conventional materials, so they require lesser thicknesses to withstand a given load.
- Composites are expensive in comparison to conventional materials, so a minimum volume of material is used, further reducing thickness as much as possible.
- The higher the volume of matrix composites, the lower the stiffness of the final composite, since the stiffness of the fiber is normally superior to the matrices'. It is also desirable to increase the moment of inertia of beams and stiffeners without increasing cross-sectional area. This is accomplished by enlarging cross section dimensions and reducing thickness.

The previous facts suggest that failure of composite stiffened panels is more likely governed by buckling. Even if our stiffened panel is made of conventional materials, the thickness of the plate is several orders of magnitude less than its width or length. The membrane stiffness for thin-walled structures is generally several orders of magnitude greater than the bending stiffness. A thin-walled structure can absorb a great deal of membrane strain energy without deforming to excess [3, 25]. It must deform much more in order to absorb an equivalent amount of bending strain energy. If the structure is loaded in such a way that most of its strain energy is in the form of membrane compression, and if there is a way that this stored-up membrane energy can be converted into bending energy, the shell may fail dramatically by buckling, as it exchanges its membrane energy for bending energy. Very large bending deflections are generally required to convert a given amount of membrane energy into bending energy [10].

3.3. Main buckling modes causing stiffened panels failure

There are two main buckling failure modes of a stiffened panel when it is subject to in-plane normal or shear loading. These modes are local buckling and global buckling (also called “Euler” buckling) [30]. There is also a special, less common case known as tripping failure. In local buckling, half waves appear on the skin between stringers. This kind of buckling normally takes place when the stringers are very stiff in comparison to the skin, and therefore the skin is not able to absorb as much load as the stiffeners before buckling. Global buckling takes place when half waves bend both skin and stringers simultaneously. This is normally the case when the stringers are too small, and do not add enough stiffness to the plate. Finally, tripping is lateral torsional buckling of a stiffener. This failure mode is normally triggered when the panel is subject to bending moments that induce initial compressive stresses in the flange of the stringer. However, it is not that usual under in-plane normal or shear loading. According to Danielson and coworkers [16], tripping of a stiffener is dependent on the torsional stiffness of the stringer. The ratio between torsional stiffness of the stringer and the torsional stiffness of the plate in bending determines whether, under axial compression, the failure mode will take the form of plate buckling or stiffener tripping.

The figures below show examples of each of the previously mentioned buckling failure modes.

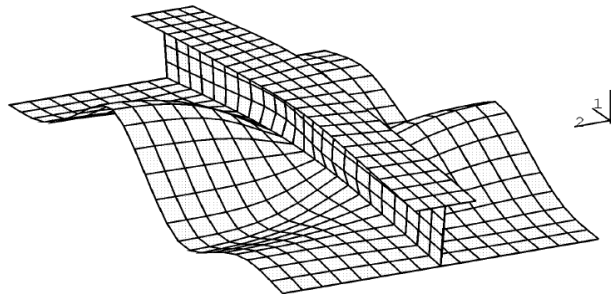


Figure 3.2: Example of local buckling -skin buckling-. Source: Figure 2 of [26].

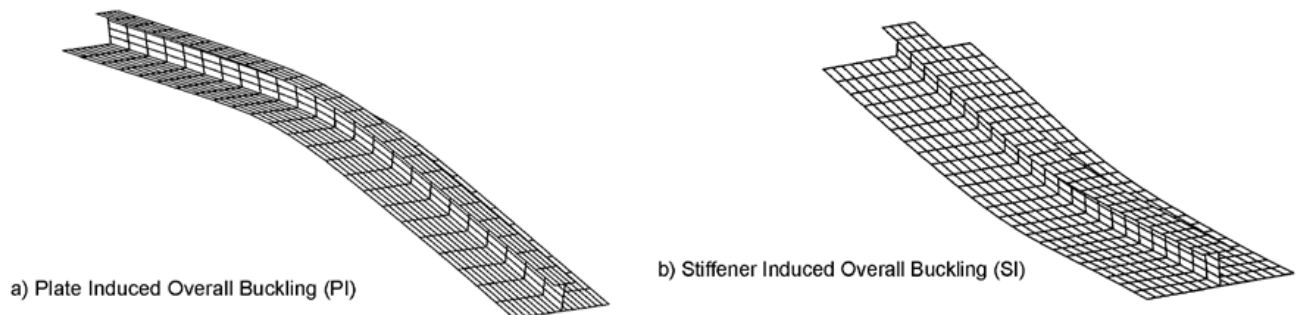


Figure 3.3: Example of global buckling -skin and stringers buckling-. Source: Figure 2 of [26].

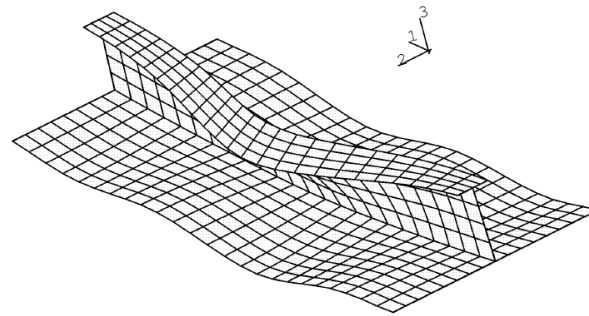


Figure 3.4: Example of tripping –stiffener torsional buckling-. Source: Figure 2 of [26].

According to Mittelstedt [39, 57] a mass-optimal configuration can be found when both local and global buckling can occur at the same critical load, whereas in [32] it is stated that when critical local and global buckling loads lie within the same range, the two modes interact, reducing critical buckling load by a significant amount. However, as verified by Grondin [26] most of the cases suggest that a failure by Euler buckling is the most favorable mode of failure since it has a more stable post-buckling behavior. In contrast, failure by tripping of a stringer is the least desirable, since it is characterized by a sudden drop in load capacity just after the peak load is reached.

In conclusion, buckling is a serious problem facing stiffened panels. Designers must understand which main modes by which a structure could fail due to loss of stability, and then find the corresponding critical loads. As mentioned earlier, the plate can experience buckling while the materials are still in the elastic range, and the severe deformations caused by buckling can lead the component to collapse. Thus, it is critical to reduce its impact during the design step, and obtain a critical load as high as possible to meet the requirements. Once the preliminary design has been set, an ordinary FEA run to the final geometry will determine whether the material will withstand the loads, and identify the regions that could suffer the most. At this point some small adjustments can be made to the model without causing large variations in the buckling load. Nevertheless, since buckling is so strongly influenced by geometry and since most sizing takes place in the design step, it is most relevant to study the buckling problem first. Consequently, the application will focus on solving the buckling problem.

Chapter 4

4. Web programming

4.1. Introduction to Object-Oriented Programming, C# and ASP.NET

Before starting development of the application, the programming language and platform must be selected. These critical decisions will determine the subsequent steps of the design and implementation processes, so it is necessary to evaluate some of the alternatives available in order to choose the most suitable tools. In this section, object-oriented programming and web development concepts and history are briefly presented to contextualize this decision.

4.2. Object-Oriented Programming (OOP)

4.2.1. History of Object-Oriented Programming

OOP began with Simulia 1 (1962) and Simulia 7 (1967) languages [46]. In the 1980s, the object-oriented language C++ was deployed from the C language. However, OOP did not truly thrive until the early 1990s thanks to the spread of C++. This language was a very powerful though complex tool. As a result, Java was released in 1991 in an attempt to overcome this complexity and simplify object-oriented programming. Java became very popular with the growth of the World Wide Web. In 2000, Microsoft developed C# from C/C++, as an OOP language considered by many to be a response to Java. Regardless of the reason, C# is a robust object-oriented language capable of accomplishing a general variety of programming tasks.

4.2.2. Object-Oriented Programming concepts

In order to understand the structure of our application and some of the decisions taken during its development, some main concepts of object-oriented programming must be outlined [35, 46]:

- Objects and classes: A class is a template used to describe an object. Once a class is defined, any amount of objects can be created within that class. Therefore, a class is a simplification of an object that could exist in real world. A class consists of two main types of components: properties and methods.
- Properties are data that define or describe an object. For example, a class named *clsisotropicmaterial* could be defined with the following properties: name, Young modulus E, Poisson coefficient nu, density, and strength. The state of an object is defined by the value of the object's properties, and so the state changes anytime a property value changes. For that reason, normally different objects have different states.
- Methods could be defined as actions that an object can perform, and are normally used to manage data contained within the object. For instance, a method of *clsisotropicmaterial* class could be `ComputeG()`, which could read the values of E and nu and compute the value of G using elasticity formulae for isotropic materials. Methods range from this simple type to much more complex actions.

There are three steps to creating an object. In order to comprehend how it works, the creation process of new object "steel" of the class *clsisotropicmaterial* will be used as an example. First a class must be defined (*clsisotropicmaterial*), which involves defining its properties and methods. Second, that class must be instantiated with the following instruction: (*clsisotropicmaterial* steel). This causes .NET to ask the operating system for 4 bytes of memory where it can store a reference (memory address) of the new object. Third, a second instruction is executed: (`steel = new clsisotropicmaterial ()`), and .NET asks whether there is enough memory to create that object. If so, a *clsisotropicmaterial* object has been

instantiated correctly. The variable “steel” will contain the memory address where the object data will be stored. As a result, the final step would be to give values to the properties of the “steel” object to change its state.

- Data Abstraction is related to the concept of encapsulation, which means that the data stored in an object cannot be directly accessed by the rest of the world. In order to access the data, some keywords (dot operator in C#) are required. Encapsulating data is an attempt to protect the data stored in an object from other parts of the program. In many cases, data inside an object can be managed by functions (methods) inside that object.
- Inheritance refers to the fact that one class can inherit the properties and methods of another class, and do not need to be defined again. By sharing properties and methods between a base class and a derived class, less code is required, and so less testing, debugging and maintenance. OOP allows significant reuse of code, making programming more efficient. To accomplish this, classes should contain the minimum amount of properties and methods required for completing the task. Therefore whenever a more complex or detailed class is required, it may be derived from the base class, inheriting all the basic properties and methods. New features may be subsequently added.
- Polymorphism is the ability to take more than one form or shape. An operation may exhibit different behaviors in different instances. The only requirement is that the signature of the method (the type and/or number of arguments it takes) must be different. For example, methods of a base class can be overloaded in a derived class, and perform a different task whether they are accessed from the derived class or the base class.

4.2.3. Advantages and disadvantages of OOP

From these main features of object-oriented programming, its main advantages and disadvantages may be summarized [46].

Advantages:

1. Improved software-development productivity: OOP is modular, separating tasks in object-based program, easily extensible as new attributes and behaviors can easily be added to objects, and can be reused within across applications.
2. Improved software maintainability: Due to modularity, a part of the system can be updated without requiring large-scale changes.
3. Faster development: Code developed in a project can be reused in future projects, as new objects can be created with small differences to existing ones. OOP provides a good framework for code libraries where supplied software components can be easily adapted and modified by the programmer.

4. Lower cost of development: Code reuse reduces development costs, as it requires less code. However, a correct design and analysis of the objects within the project must be achieved.
5. Higher quality: Faster and lower-cost development frees more resources and time for verification of the software.

Disadvantages:

1. Complexity: OOP takes time to master. Some programming techniques such as inheritance and polymorphism can be initially challenging to comprehend.
2. Larger program size: These programs typically require more lines of code than procedural programs.
3. Slower programs: As a consequence of program size, there are more instructions to be executed.
4. Suitability issues: Some problems may be less efficient if developed with this approach.

4.3. Web development

4.3.1. History of web development

The internet was invented in the late 1960s in an attempt to create a highly robust information network that could withstand the loss of several computers without preventing the others from communicating. The early version was solely available for educational and defense institutions. However, it soon became a powerful tool to share information among researchers around the world. In the early 1990s, modems capable of communicating via phone lines appeared and the internet began to open to commercial users. In 1993 the first HTML browser was created, ushering in a revolution.

Early web development platforms did not always scale well, as popular websites would struggle when facing too many simultaneous users and provided only simple, low-level features. More advanced features such as reading a database required pages of original code, making the process tedious and error-prone.

To solve these problems, Microsoft created higher-level development platforms, ASP and later ASP.NET [36], which allow programmers to pass over low-level implementation concerns by using a significant number of sophisticated tools.

There are two main approaches making an interactive web page: server-side and client-side programming approaches. ASP.NET was designed principally as a server-side programming platform, which means that all code runs on the web server, and after all code finishes running, the web server sends the user the final result. In contrast, sites may use

pieces of JavaScript code or plug-ins such as Adobe Flash, which are downloaded to the client's computer and executed from there. The main advantages of server-side programming are the isolation of client-side from access server side resources such as databases, security advantages for end users who can see client-side code and modify it, and a broader client base including web-enabled devices such as smartphones or tablets that do not support client-side programming platforms such as Flash.

As a consequence, server-side platforms are more suited to sites that display and manage large amount of data such as e-commerce hub or business sites. Client-side platforms have advantages for developers interested in building a real-time browser-based game, or fast, responsive, dynamic pages.

4.3.2. The “.NET” framework

The “.NET” framework consists of several technologies [12, 35]:

- The “.NET” languages: These include Visual Basic, C#, F# and C++. Since all these languages are compiled into another lower-level language (Common Intermediate Language, CIL) before being executed, there is no real preference as to which of the previous languages is used for programming. The DLL or EXE file that CLR will execute is precisely IL code, so the CLR makes no distinction between different languages, also called language integration.
- The Common Language Runtime (CLR): This is the engine that executes all “.NET” programs and provides some automatic services such as memory management and optimization. CLR achieves deep language integration, fewer errors than lower-level languages such as C++, and side-by-side execution. However, it also has some drawbacks such as a reduction in performance for a few performance-critical high-workload applications such as real-time games, and code transparency. The latter is not an issue for ASP.NET apps since they are not distributed but hosted on a secure web server.
- The “.NET” Framework class library: Thousands of prebuilt features are collected and can be used in the applications.
- ASP.NET: This engine hosts the web applications that are created with “.NET” and supports almost any feature from the “.NET” Framework class library.
- Visual Studio: This optional development tool includes some features that make the creation process easier, faster and less error-prone. Visual Studio assists in page design and debugging, detecting errors automatically, and speeding up the development process of the application with IntelliSense. This feature recognizes objects and parameters so that the developer does not have to type the whole name.

4.4. Programming language selection

The aim of this project is to create an engineering tool that expedites the design of stiffened panels made of composite material. That tool will be implemented in a web environment, and will require high-level features including databases and users authentication. Object-oriented programming will be used due to its aforementioned advantages, and will also allow for future expansion of this tool. According to the different alternatives that have been described during this section, the development application used to create this tool is Visual Studio 2010. Based on the needs of this design tool, a server-side programming platform is the best option and so ASP.NET will be used. Code-behind will be written in C# thanks to its higher-level character and its similarities to C syntax.

Chapter 5

5. The Design tool

During the design step of a stiffened panel a lot of variables come into play, which leads to a decision-making process for the designer that influences importantly the results. Normally some of the dimensions of the panel are fixed as it may be a part of a more complex set, but several times there is some freedom such as the number of stiffeners to use, their shape or some of their dimensions. Another critical point is deciding the material to use both for the skin and the stringers, which again provides uncountable possibilities. The expertise of the designer will considerably reduce the possible designs, but there will be still several options available.

The purpose of this application is to help the designer during this process. And the main goal is to reduce time during the design step, allowing the user to change some features very easily as well as testing their impact and performance. It will neither be necessary to create a different model from zero each time the designer wants to change a dimension, nor to waste a lot of time assigning materials. This application will provide an intuitive and effective way of managing all these variables and it will provide both the buckling critical load and the first mode displacements of the panel, so that the user can compare among different solutions and choose the one that better fits the requirements. It should be noticed that during the design step we need to discard different alternatives so that a deeper study is just carried out to one or two designs. So in this step the accuracy both in the input and the output data is not that important as it will become in the analysis of the final design.

The application consists of several components, each of which plays an important role:

- The mesh generation tool, which takes the geometry variables and creates panel geometry and mesh.
- The Load and BC manager tool, which converts the information supplied by the user into a form that is managed by a Finite Element solver.
- The problem builder tool, which gathers all the relevant information of the problem so it can be solved. This information includes the mesh, materials, boundary conditions and loads. This is the core of the program, and it will manage most of the process from the user's input data until the final results are returned.
- The solver, which takes the information provided by the previous component and solves the buckling problem. This solver already exists and was not developed in this project. However, certain improvements were required in order to achieve the best results possible.
- The visualization tool, or at least the tool in charge of presenting the data to enable a third party program such as gmsht [21] or Abaqus® to visualize the results.

In the following sections each of these components will be explained in further detail so that a better understanding of the application is possible. It is important to clarify that the different tools are not totally independent, but they are partially overlapped.

5.1. Coordinate Systems (CSYS) utilized

Before beginning with the explanation of each class, it is important to clarify the Coordinate System (CSYS) used within the application. Both the global CSYS and each element local CSYS gain relevance when it comes to the definition of material properties and the application of loads and BC.

5.1.1. Default CSYS of the mesh generator

As far as the mesh generator tool and the BC and Load manager are concerned, the coordinate system used by default is the following:

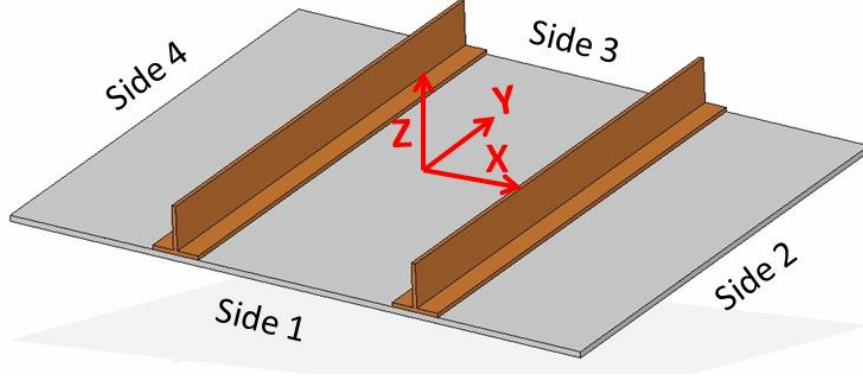


Figure 5.1: Default Coordinate System employed by our tools, different from the one deployed within CADEC environment. Created using Solid Edge®.

The X axis is oriented parallel to sides 1 and 3, along the width of the panel. The Y axis is perpendicular to X and contained in the skin plane, and oriented so that Z axis is perpendicular to the plate pointing upwards.

The problem behind this coordinate system is that the primary direction (X axis) is not parallel to the extrude direction. Normally, material properties are specified so that material direction 1 is parallel to the extrude directions, so that fibers oriented with 0° are parallel to the stiffeners. Nevertheless, this issue could be solved just by defining material properties accordingly, that is adding 90° degrees to the default orientation. The real problem of this CSYS is related to the solver.

5.1.2. Calculation of local CSYS for each element by the solver

How does the solver define the local CSYS for each element from the global CSYS? Another orthogonal CSYS is created for each element [47], which assumes that the element is flat. For this reason, panel curvature should not be excessive to ensure accurate results.

The surface of the element is defined by the vector $\vec{r} = X\vec{i} + Y\vec{j} + Z\vec{k}$. The values of X, Y and Z are, respectively:

$$X = \sum_{i=1}^N N_i X_i$$

$$Y = \sum_{i=1}^N N_i Y_i$$

$$Z = \sum_{i=1}^N N_i Z_i$$

Where:

- N is the number of nodes of that element, in our case $N=9$.
- N_i stands for the shape functions of that element.

The shape functions are dependent on two parameters ξ and η so that $N = f(\xi, \eta)$. Since the solver assumes that elements are flat, the normal vector \bar{n}_z is computed exclusively in one node per element. Then, the normal vector calculated in a specific point denoted with subscript zero is:

$$\bar{n}_z = \left| \frac{\partial \bar{r}}{\partial \xi} \right|_0 \times \left| \frac{\partial \bar{r}}{\partial \eta} \right|_0$$

Once the normal vector is defined, the in-plane axes are computed. The process followed by the solver to compute local \bar{n}_x and \bar{n}_y axes for each element follows. First, the solver tries to compute the secondary axis as $\bar{n}_y = \bar{n}_z \times \vec{i}$, where \vec{i} is the unitary vector in direction of X global axis. Then, the primary axis is computed as $\bar{n}_x = \bar{n}_y \times \bar{n}_z$. Finally, all vectors are normalized so they have unitary module. The previous methodology is valid except when \bar{n}_z is parallel to \vec{i} . In the case of our panel, this situation occurs whenever an element is in the plane YZ (vertical web elements). Under such specific circumstances, the local in-plane axes are computed using an alternative procedure. First, the primary local axis of the element is computed as $\bar{n}_x = \vec{j}$, so that it is parallel to the global Y direction. Then, the local Y axis is computed by $\bar{n}_y = \bar{n}_z \times \bar{n}_x$. After that, vectors are normalized as in the previous case.

So the problem with the default CSYS is obvious once the previous methodology is understood. The following image shows schematically the local directions in several elements of a panel. The figure of the top represents how one would expect the local CSYS to be, and figure at the bottom illustrate how vertical elements are assigned an orientation which is 90 degrees rotated with respect to the expected one due to the previous algorithm.

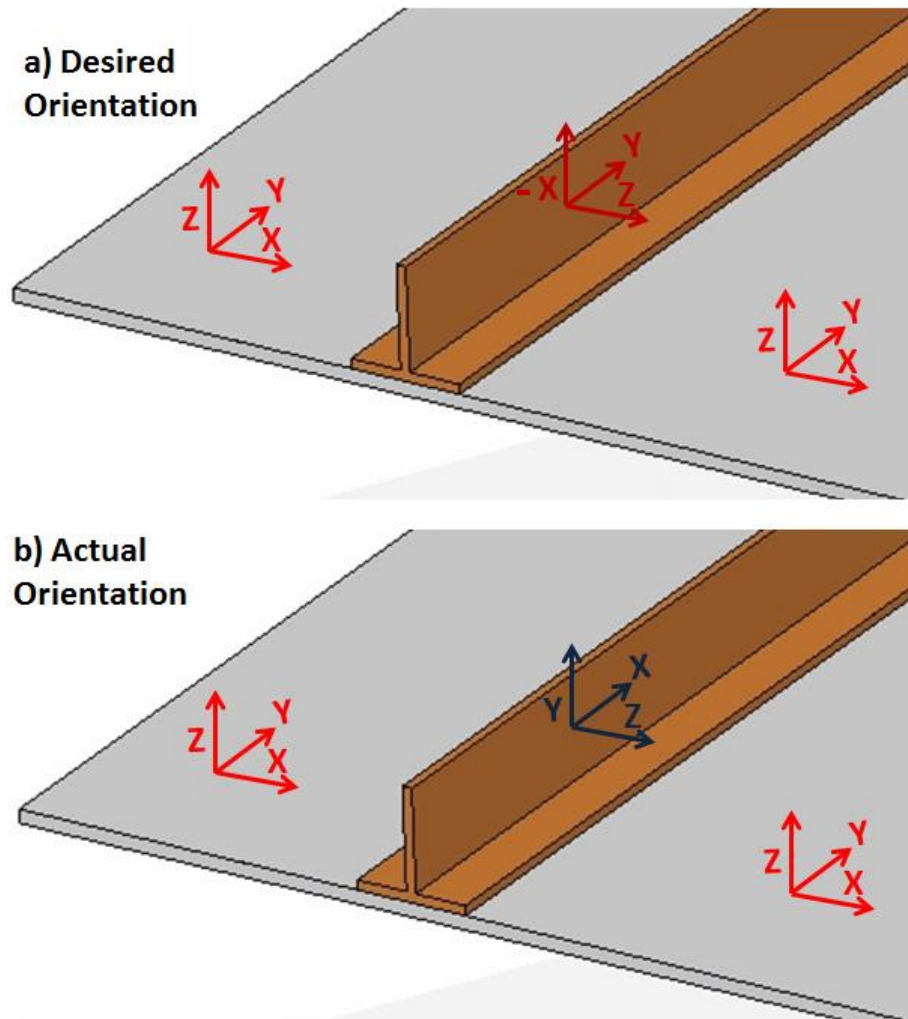


Figure 5.2: Coordinate System inconsistency when using default CSYS.

This would cause the user important difficulties since material orientations for the vertical elements must be defined with additional rotation. Therefore, there is a strong motivation to achieve the desired local coordinate system for every element.

5.1.3. Final CSYS used by CADEC

Since the previous orientation could be valid in case the solver uses the Y axis parallel to extrude direction, modifying the whole code was not an interesting option. In fact, the most useful improvement would be to add some functionality to the code so that the axis coinciding with the extrude direction could be specified. For that purpose, the following modifications were performed:

- *SPmshBuilder.cs* includes a new method that allows to rotate the panel a certain angle around the vector (n_x, n_y, n_z) so that the method computes the

new coordinates of the nodes. In particular, the panel will be rotated around the Z axis -90° to achieve the CSYS orientation shown in the next figure.

- *INPUTdata.cs* requires an additional argument to specify the axis which is parallel to the extrude direction, which could be X or Y. It is important to impose the correct orientation when calculating loads and boundary conditions.

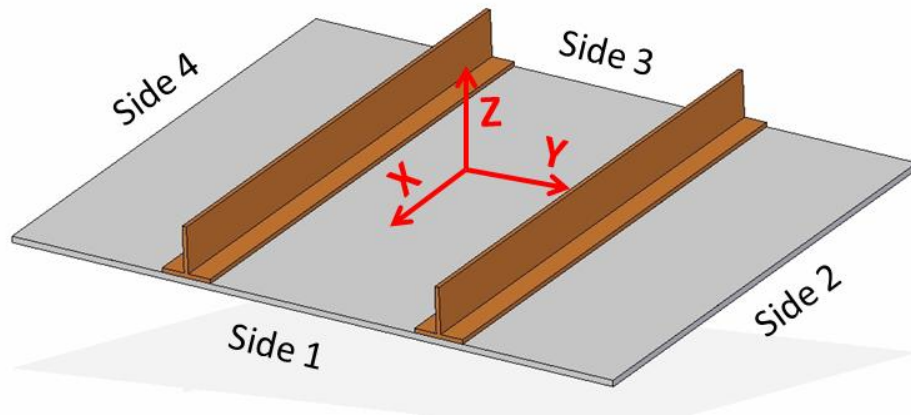


Figure 5.3: Final CSYS used in CADEC. Created using Solid Edge®.

CADEC [2], as it will be explained in another chapter, is the web environment in which our tool will be implemented. Therefore, when implementing the tool it is important to apply the rotation of the panel as well as specifying that change to the BC and load manager. For the final user, the previous CSYS is the one that will always be used in the final application. In this way, the problem with the default CSYS is solved. However, it is important for the programmer's point of view to understand that it is possible to specify the orientation in case in the future this application communicates with a different solver that has a different algorithm for the definition of local CSYS.

The following figure shows how, under the new CSYS, the local coordinate systems are computed:

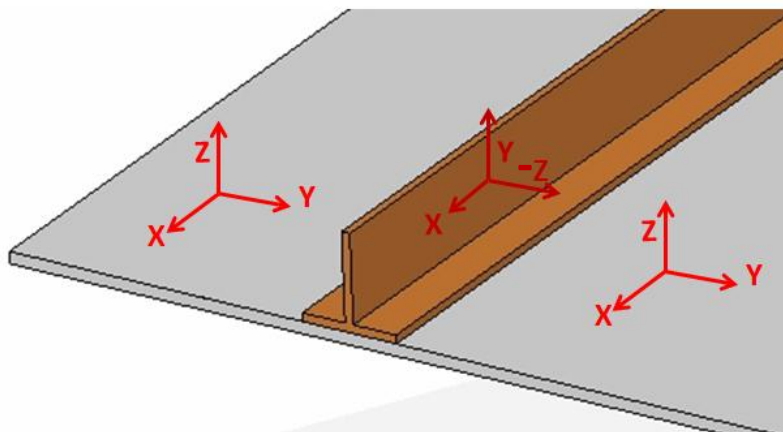


Figure 5.4: Local elements CSYS in CADEC. Created using Solid Edge®.

5.2. The Mesh Generation tool

The Mesh Generation tool acquires the geometric parameters introduced by the user and processes them to create the mesh of the panel. Meshing is the action of dividing the body into smaller entities called elements in order to solve the problem. Since gmsh is no longer utilized for this purpose, a complete new tool was developed from scratch. Therefore, the following classes are defined: *FlatRecGeo.cs*, *CurvedRecGeo.cs*, *MeshParameters.cs* and *SPMshBuilder.cs*.

The meshing process is the following. First, the panel is divided into different subparts, each of which could be a flat or concave parallelogram. Then, each subpart is created from left side of the panel to right, and assembled with the previously created subparts until the whole panel has been created.

The first two classes (*FlatRecGeo.cs* and *CurvedRecGeo.cs*) are those that actually mesh a parallelogram. In other words, these classes could be used for meshing different geometries, since they have not been conceived specifically for panels. Provided that the geometry intended to be meshed can be divided into parallelograms, it is possible to use these classes.

The third class (*MeshParameters.cs*) is defined specifically for stiffened panel, since it computes the mesh refinement level based on the dimensions and on elements location within the panel. For instance, the element size will be different for a web than for a skin.

Finally, the last class is actually the brain that creates the panel geometry and mesh, relying on the other three classes to achieve that.

Next the content of these classes will be explained. A further explanation of the possible geometries that can be created by the tool can be found in the section 5.3: *Problem Builder Tool* in this chapter, as well as in *Chapter 7: Implementation of the application*.

5.2.1. Mesher of flat surfaces: FlatRecGeo.cs class

A. Aim and arguments

The aim of this class is to create low level geometry. As previously explained, the final piece is created by the combination of smaller parallelograms, which can be flat or curved surfaces. Each object of this class will create the nodes and elements that define a geometric space limited by four straight edges, parallel two by two. In this case, the class meshes flat surfaces.

The following arguments are required to create an instance of this class: a name, three points defined by three coordinates in cartesian CSYS (corresponding to three vertices of the parallelogram), number of elements desired on two edges non-parallel edges, type of element (*quad4* - first order quadrangular element-, *quad9* - second order quadrangular element-, or *tria3* - first order triangular element) and information about the ID of the last element and node that were created in the global geometry.

Next image shows an example of one of these parallelograms meshed using the three different types of elements. Notice that the mesh containing triangular elements is not a proper mesh due to elements arrangement: such a structured mesh can lead to accuracy problems in results. However, this option was implemented both due to its ease of development from the previous meshes and to add more functionality to the mesh generator tool. Nevertheless, it is important to realize that the final application (CADEC) will use just the second order quadrangular element since it provides the best results.

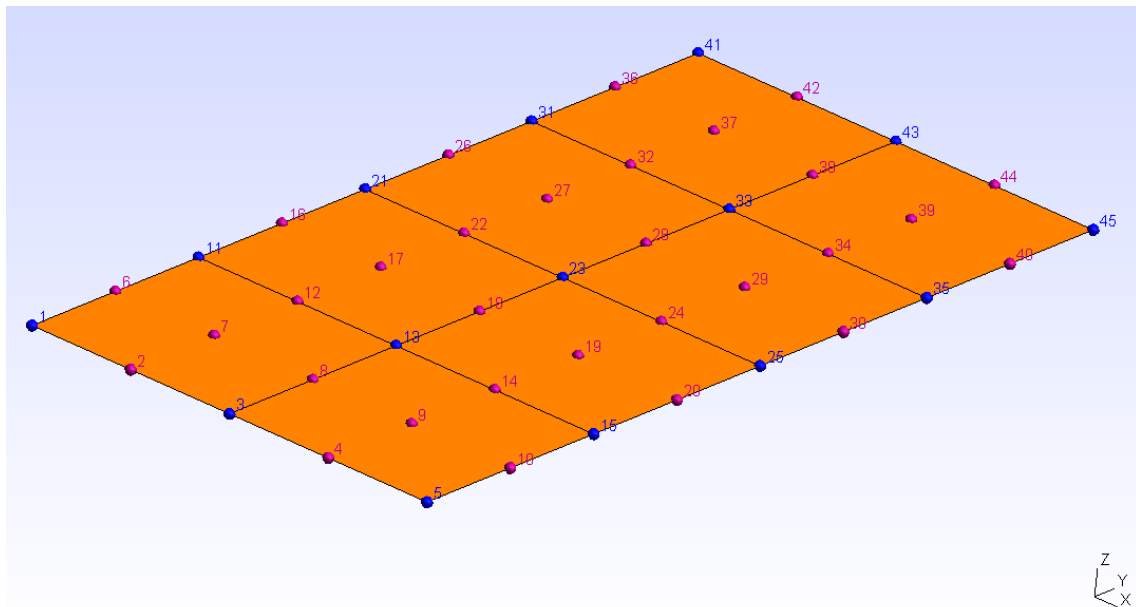


Figure 5.5: Flat parallelogram meshed with second order quadrangular elements using CADEC. Image visualized using gmsh [21].

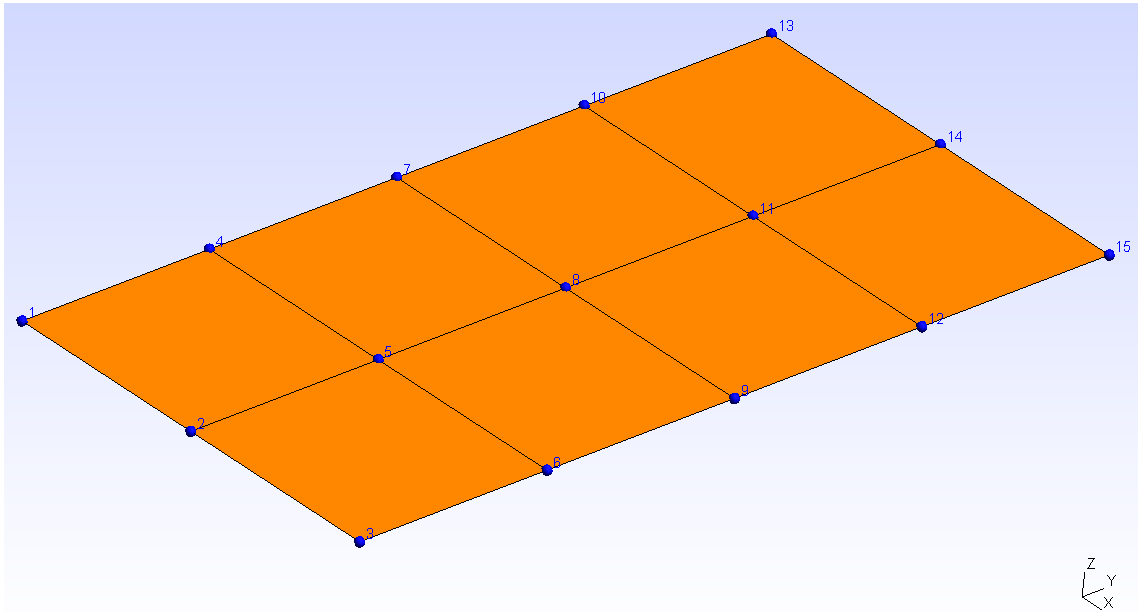


Figure 5.6: Flat parallelogram meshed with first order quadrangular elements using CADEC. Image visualized using gmsh [21].

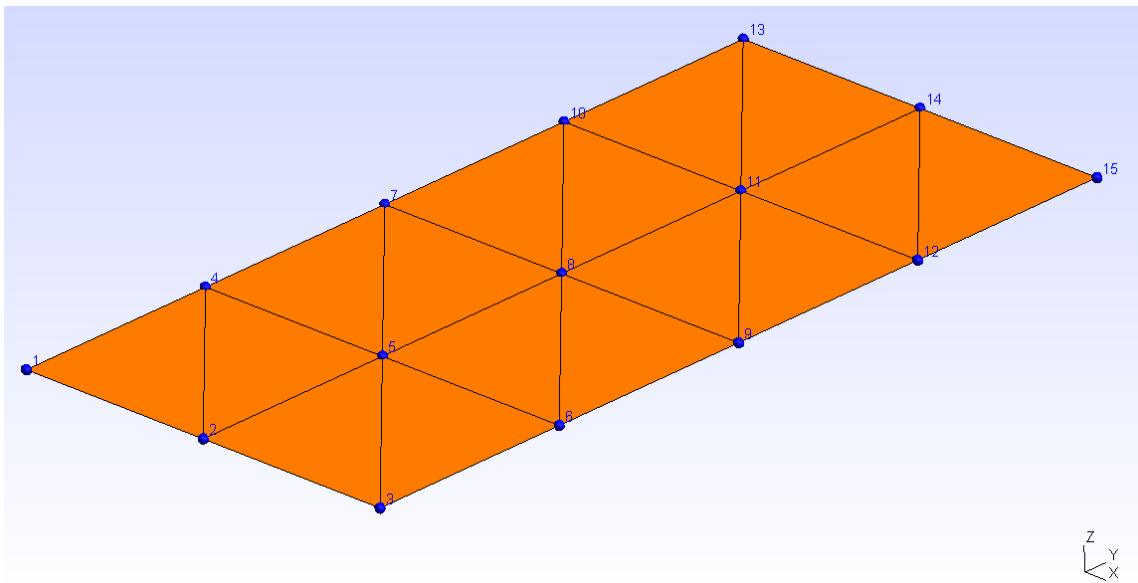


Figure 5.7: Flat parallelogram meshed with first order triangular elements using CADEC. Image visualized using gmsh [21].

B. Methods

Helper Methods:

The class contains several internal methods which support the main methods. These methods deal with coordinate conversion from a cartesian CSYS to a cylindrical CSYS, rotation of one or several points around a main axis or even a particular method to compute the coordinates of key points for the creation of the omega profile in a curved panel. Although this class just creates flat surfaces, previous methods are necessary to position a new sub-mesh in the geometry that has already been created. A deeper overview of such methods can be found in the program code in case new stringer profiles or a different geometry is required. However, for the purpose of this document, these methods are not considered relevant for the understanding of the mesh generation process.

Main Methods:

There are six main methods. The first three are responsible for node generation, whereas the remaining three are in charge of element creation: connectivity definition. Due to their relevance, they will be explained in greater detail.

- *AddNodesIndep* method generates nodes independently, which means that none of the sides is connected to existing geometry when the subpart is created. Therefore, nodes can be added directly from last ID with no concern about previous nodes. The methodology followed in node generation is to divide the surface into small rectangular elements, whose size would be determined by the number of elements required. Depending on the type of element, its size and the seeds number are adjusted. Based on previous information, a grid is created, and by going through it nodes' ID and coordinates are established.

From another point of view, let us consider the image below. Let us forget about the elements since this method just creates the nodes, being it possible to use them to define any of the three types of elements that were explained before. In order to mesh this parallelogram, the coordinates of geometric vertices located at nodes 1, 5 and 41 are passed as arguments when an object of this class is instantiated. From coordinates of vertices two and three, coordinates of vertex four are computed, which coincides with node 45. The position of these points within the parallelogram defines the dimensions of the parallelogram as well as the sides labeling: edge 1 is defined by points 1 and 5, edge 2 by points 5 and 45, etc. Then, the seeding process starts. Notice that edge 1 is seeded with equidistant nodes following the red arrow, and then they are extruded following edge 4 direction following the blue arrow, copying them at several intervals till the end is reached, which corresponds to nodes defining edge 3. The distance between these parallel rows of nodes depends on the seeding of edge 4.

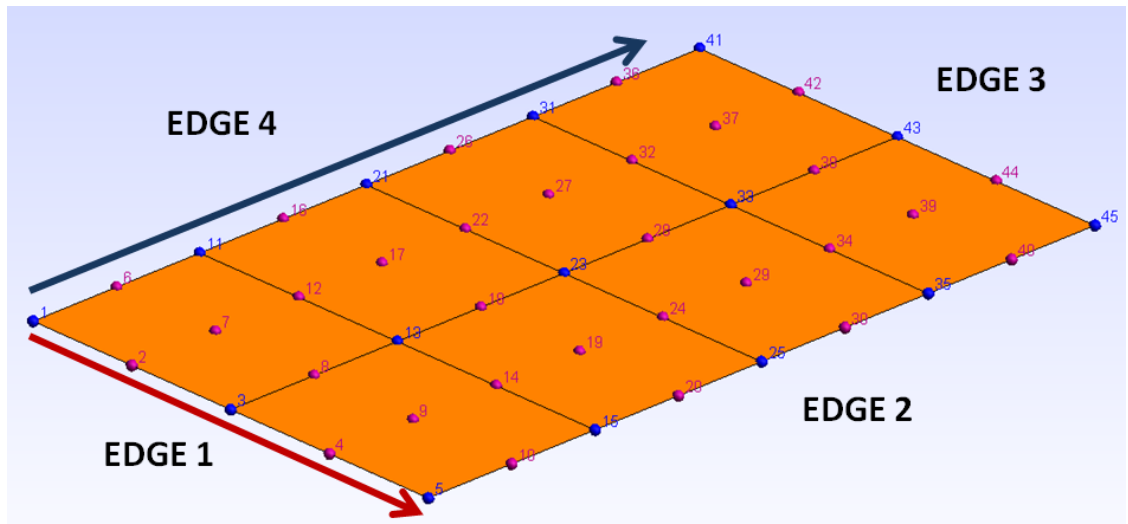


Figure 5.8: Meshing sequence for a flat parallelogram -nodes generation-.

This method requires no extra arguments, and it gives back the ID of the last node, which will be used for the creation of next surface mesh.

- *AddNodesdep4side* method generates nodes dependently. In this case, the new mesh will be added to a previous one, resulting in some nodes shared. Then, the edge 4 of the new mesh will be connected to previous geometry. It may be obvious, but for this operation to succeed the number of nodes on the edge and their position must be the same for both surfaces, and they must have the same ID. As a result, the seeds number on that direction is given by the common edge. This method is then in charge of reusing the already created nodes and adding the new ones. In order to create the nodes, it requires as additional input a pointer to an array containing information of the common edge. When it finishes, the ID of the last node created is returned.

The next image shows this process. The parallelogram contained within black dashed lines has been created with this method and added to the geometry on the left. Notice how the ID of nodes corresponding to edge two (highlighted with a red box) come from the previous geometry, and how new nodes are added continuing from last node ID (starting with node 46).

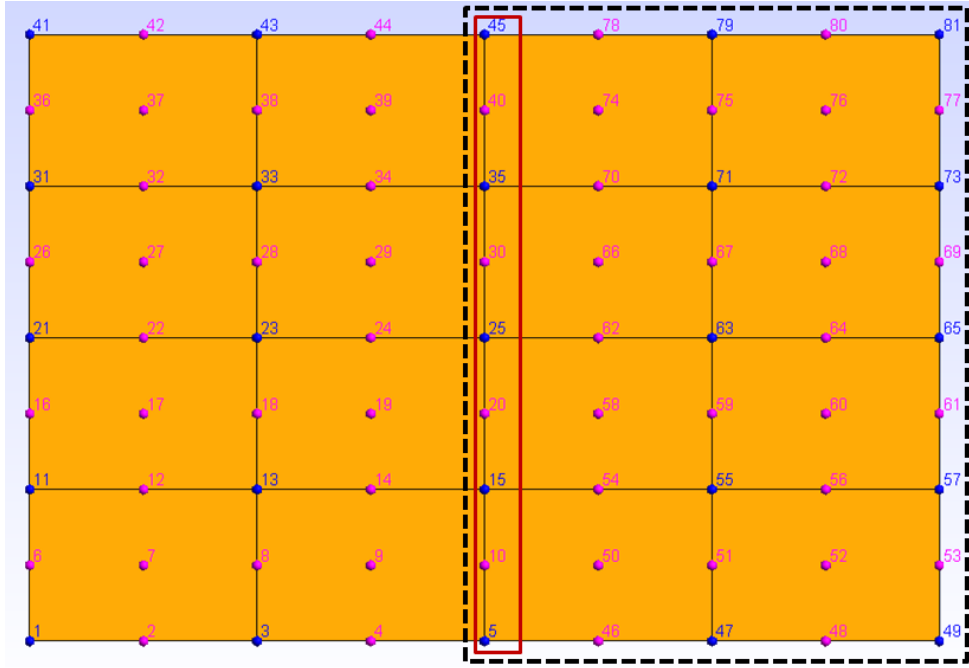


Figure 5.9: Joint representation of two subparts sharing a common edge.

- *AddNodesdep2_4sides* method generates nodes dependently. In order to create some profiles, specially closed ones, some meshes must be added between two edges that already exist. This method provides this service, being the only requirement that both edges be parallel and contain same number of nodes. Therefore, both edges are required as input parameters, and once the method finishes, the ID of the last node created is returned.

Considering the panel geometry with which our application is expected to deal, these three methods are sufficient. However, from the perspective of future development of the tool, it could be interesting to add new methods to improve connectivity between different sub-meshes. For example, enabling assembly of sub-meshes with two non-parallel common edges, or generating new mesh with more than two common edges. This could be achieved with little modification of the code of previous methods. Unfortunately, that is out of scope and it will not be studied in this report.

At present, nodes of a sub-mesh have been already created by means of one of the previous methods. However, they are just points with ID and coordinates. In order to become elements, the connectivity must be defined. There are three types of elements, as they were discussed previously. Due to differences in shape, number of nodes, and nodes sequence order for each element, three different methods were necessary to accomplish that task:

- *AddQuad4Elem* method creates the elements of the mesh. As its name suggests, it will mesh with first order quadrangular elements. This job is done by going over the grid and adding the ID of corresponding nodes in the correct order. They require no additional input data, as the grid of nodes was stored in an array as a property of the

object and, therefore, it is completely accessible. Once it is finished, it returns the ID of last element created.

The next image shows this process. The elements are created following blue arrow first, and then moving on red arrow direction. The ID of each element is represented in green.

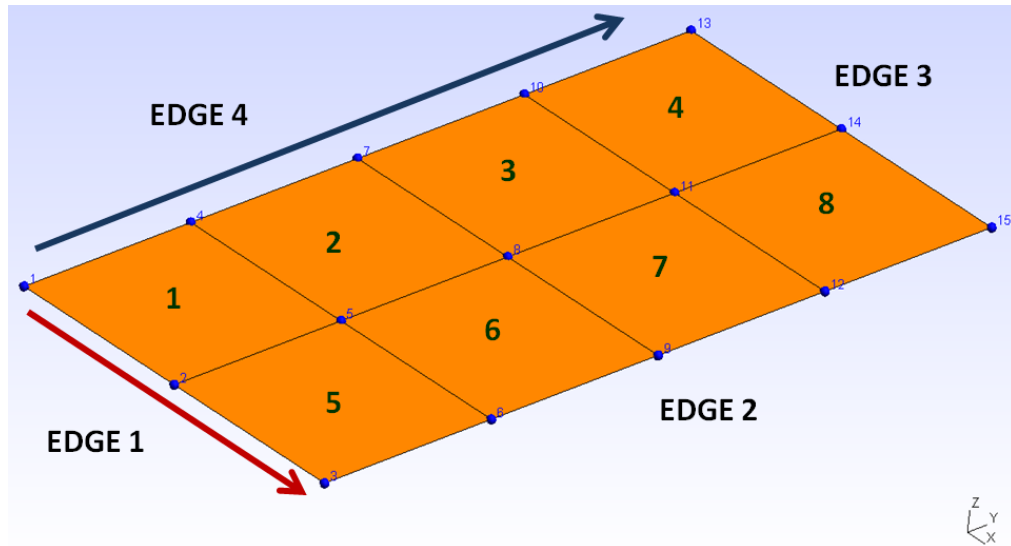


Figure 5.10: Meshing sequence for a flat parallelogram –first order quadrangular elements generation-.

- *AddQuad9Elem* method creates second order quadrangular elements for the mesh. The way it goes through the grid is quite similar to previous one, but since the nine nodes need to be assigned in a specific order it requires some adjustments. Again, no additional arguments are required, and it returns the ID of last element created. An example of mesh with these elements can be found in the image below.

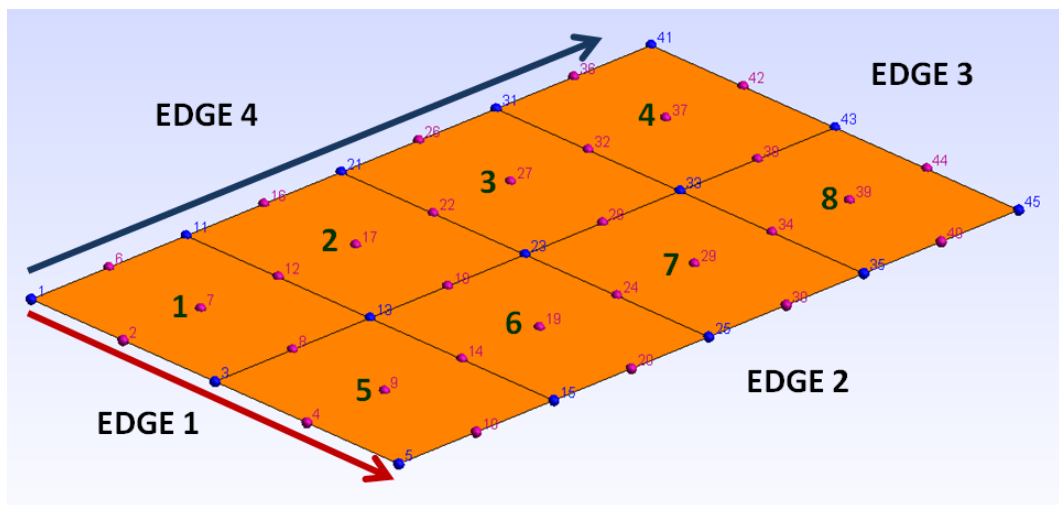


Figure 5.11: Meshing sequence for a flat parallelogram –second order quadrangular elements generation-.

- *AddTria3Elem* method creates first order rectangular elements for the mesh. The way it goes through the grid is quite similar to previous ones. Triangles are created by dividing *quad4* elements in two *tria3* elements. Better methodologies to mesh with triangular elements could have been considered, but according to our goal this is simple and effective. Besides, analyses carried out with CADEC will use exclusively second order quadrangular elements, since they provide best results with less computational cost: fewer elements are required to provide similar results for a given mesh of any of the other two types of elements. Again, no additional arguments are required, and it returns the ID of last element created. An example of mesh with these elements can be found in the image below.

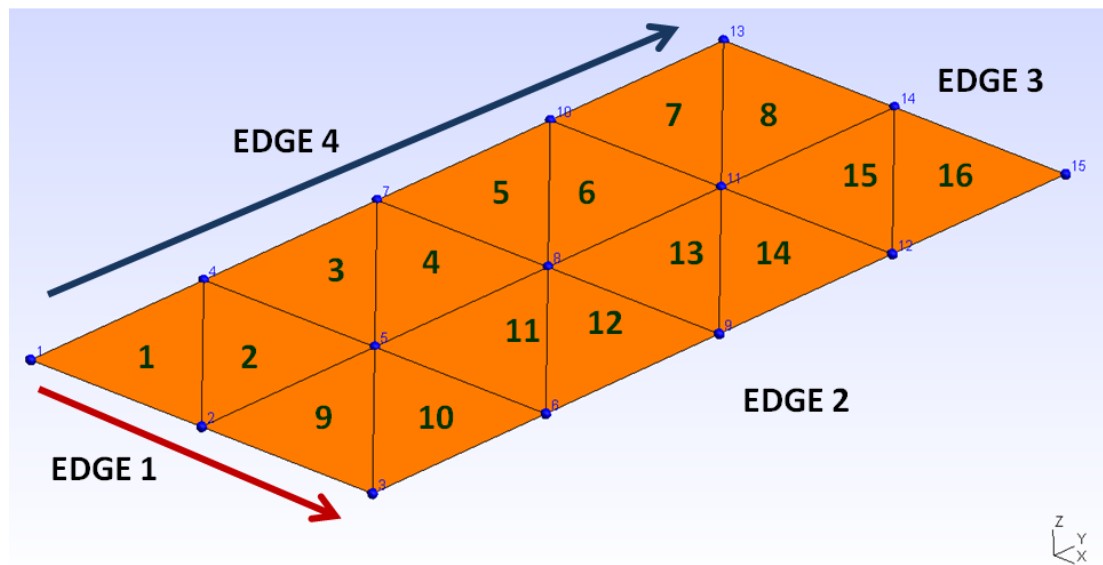


Figure 5.12: Meshing sequence for a flat parallelogram –first order triangular elements generation-.

5.2.2. Mesher of concave surfaces: CurvedRecGeo.cs class

A. Aim and arguments

The aim of this class is to create low level geometry. Each object of this class will create the nodes and elements defining a geometric space limited by four edges, parallel two by two. In this case the parallelograms are, in fact, rectangular curved surfaces. The next image shows a mesh generated by this class using second order quadrangular elements.

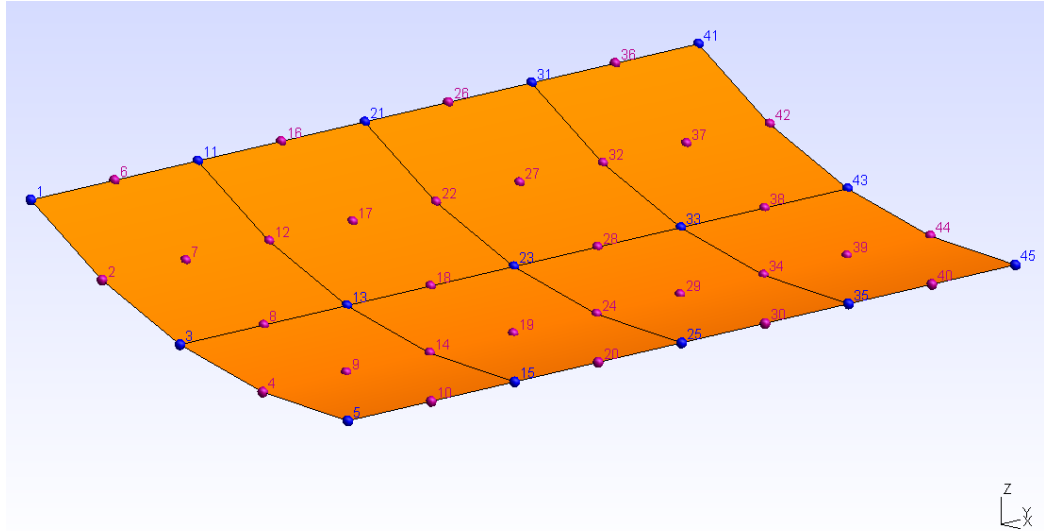


Figure 5.13: Curved parallelogram meshed with our application. Image visualized using gmsh [21].

The arguments required to create an instance of this class are very similar to the ones of *FlatRecgeo.cs*: a name, two points defined by three coordinates in Cartesian CSYS (which correspond with two vertices defining the edge 4 of the parallelogram), radius of curvature, arc length (length of the curved edges 1 and 3), number of elements desired on two non-parallel edges, type of element (*quad4* - first order quadrangular element-, *quad9* - second order quadrangular element-, or *tria3* - first order triangular element) and information about the ID of the last element and node that were created in the global geometry.

B. Methods

Helper Methods:

The class contains three internal methods supporting the main methods. First two methods deal with coordinate conversion from a cartesian CSYS to a cylindrical CSYS and vice-versa, whereas the third method computes the coordinates of the third vertex, which are the coordinates of the point at the tip of the arc.

Main Methods:

There are six main methods. The first three are responsible for node generation, and the remaining three are in charge of element creation (connectivity definition). The only difference between *FlatRecgeo.cs* and *CurvedRecgeo.cs* is the shape of the surface, and so the way the nodes are placed in the space. For that reason, the last three methods, which are in charge of creating elements connectivity are exactly the same as in the previous class. No further explanation will be provided regarding these methods. However, how nodes are generated deserves additional comments.

AddNodesIndepArc, *AddNodesdep4sideArc* and *AddNodesdep2_4sidesArc* methods accomplish the same task as their counterparts in the previous class. The first adds the nodes independently, whereas the other two have one or two common edges with previous geometry. The way nodes are generated is in accordance with the approach followed by their counterparts: nodes are created following a curved edge (not straight as previously), and the distance between nodes depends on the seeding set. Afterwards, this row of nodes is extruded following the direction of edge 4 until edge 3 is reached. Arguments as well as returned values are the same as for their counterparts.

5.2.3. Seeding of panel mesh: MeshParameters.cs class

A. Aim and arguments

This is clearly a dispensable class, but it provides an outstanding help to establish how the panel edges are going to be seeded. As a result, its aim is not other than automate the seeding of the panel based on the geometry and the proportion between some dimensions.

Then, arguments required to create an instance of this class are: all geometric values of the final panel (width, length, type of stringer, top and bottom flange widths, web width, number of stringers and pitch) as well as two additional parameters named *propmesh* and *propBmesh*. The first parameter determines the number of elements in “A” direction desired for the widest skin part of the panel, which could be an intermediate skin or a lateral skin. The second parameter determines the maximum allowable aspect ratio of elements. For instance, a value of two means that the number of elements in “B” direction (along panel length) will be the minimum satisfying that the length of an element is less than two times its width.

After carrying out some convergence analysis for several panels, and in order to find an equilibrium between the accuracy of the solution and time-consumption, these values will be set to *propmesh*=3 and *propBmesh*=2 by default. Regarding the stringers, flanges will contain just one element in “A” direction, and web requires two elements in order to absorb correctly shear loads. Finally, in case the panel has no stiffener, the number of divisions will be rounded to four in order to achieve better results.

B. Methods

No methods are required in this class. When calling the constructor to create an object, all the calculations and comparisons are performed. *Propmesh* variable and the previously explained ratio allow the constructor to compute all seeds in edges 1 or 3. Nevertheless, no data was required regarding seeds in the extrusion direction, which is parallel to sides 2 and 4 of the panel. This can be set in two ways: manually or automatically. The former would be modifying the property value to the number of elements desired in that direction just after creating the object. The latter would be calculated by the class when creating the object. In this way, the code compares the width of elements belonging to lateral

or intermediate skins to establish the seeding value, so that the maximum aspect ratio of elements of the skin is two.

It should be noticed that size of stringer elements are not considered because normally their size is quite small in comparison to skin elements. Therefore, in order to have square elements in there, their size would be so reduced that it would lead to a very fine mesh, increasing computational time excessively, preventing it from being effective for a web application. However, as shown in chapter 9, this does not prevent the tool to obtain surprising accurate results.

5.3. The Problem Builder tool

The Problem Builder tool is the core of the application, being in charge of creating the whole panel by means of the rest of the tools, specially the mesh generator tool. Whereas the classes forming the mesh generator tool are basic to generate the mesh, they are not specific for the geometry of our panel, since they just mesh parallelograms. However, the following class contains all the necessary methods to generate each of the parts of the profile (skin, stringers,...) and their assembly. This tool actually computes the dimensions of each of the subparts based on the user input data and generates the panel part by part until the mesh of the model is completed.

This task is accomplished by the following class: *SPMshBuilder.cs*. However, it depends on the other classes that are part of the mesh generator tool: *FlatRecGeo.cs* and *CurvedRecGeo.cs*.

5.3.1. Panel mesh generator: *SPmshBuilder.cs* class

A. Aim and arguments

This is the main class of our mesh generator. *SPmshBuilder.cs* (Stiffened Panel mesh Builder) is responsible for creating the final mesh based on several inputs, most of them defined directly by the user. It manages the order in which each sub-mesh must be created and added to the previous partial mesh, and stores every node and element as well as the entity to which they belong, required for the application of boundary conditions and loads. It requires *FlatRecgeo.cs* and *CurvedRecgeo.cs* classes in order to create the sub-meshes.

Most arguments required are related to final geometry and mesh configuration: a name, total width and length of the panel, radius (zero value represents a flat panel), number of stringers and type, pitch, stringer dimensions and several meshing parameters such as element type or number of elements in “A” or “B” directions for the skin and the stringers.

B. Methods

Helper methods:

Despite not creating nodes or elements, two methods are required to perform some initial calculations.

- *ComputeFx* method obtains the width of lateral skins, which is the distance from edges 2 or 4 to the first or last stringer respectively. It is necessary for internal computations, but it lacks relevance for final user. This dimension can be identified in the following image, as well as directions “A” and “B” of the panel.

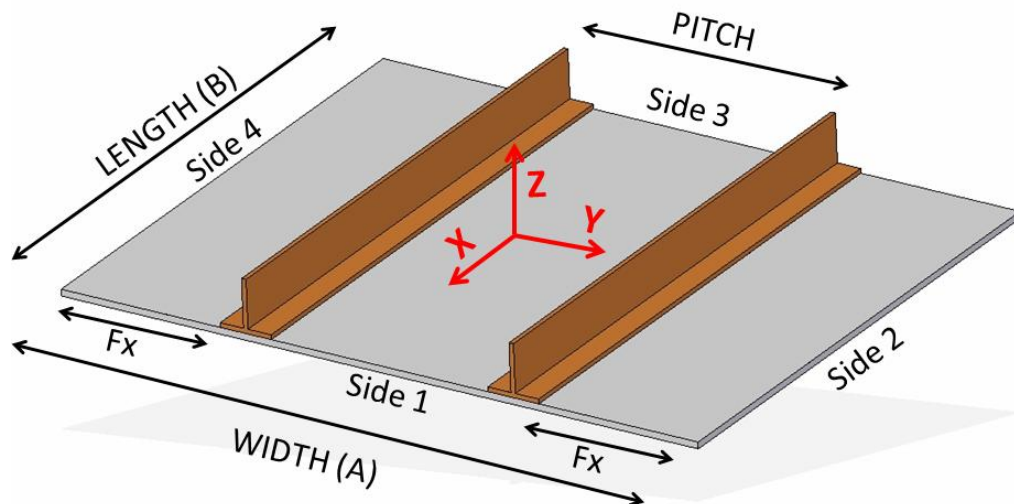


Figure 5.14: Definition of some basic parameters required to create a stiffened panel. Created using Solid Edge®.

- *InitializeData* method is in charge of predicting the number of nodes and elements that are going to be created in the final panel in order to define the size of the different variables that will store the information.

Both helper methods are called directly from the constructor, but it was decided to separate the code for simplicity and reading ease.

There is a third helper method called *PrintMSHData*, which should be called after the final panel is already created. Its aim is to store in a string the information necessary to visualize the mesh with the freeware gmsh [21]. The structure of the .msh file can be found in *Appendix A*.

Main methods:

There are nine important methods in charge of creating specific parts of the final panel, and an additional method which manages all of them to create the final mesh. In case a different profile is required, it would be necessary to create an additional method for creating the new geometry of that stringer, following the same structure as the others. Additionally, some code would be added to the main method too so the new geometry is assembled whenever a stringer of that type has to be placed.

- *CreateLeftPanel* method: geometry is built from left (negative X direction) to right (positive X direction). Therefore, this is the first method that must be called. It creates the skin from edge 4 of the panel to the bottom flange of the first stringer, following the next procedure:
 1. Check whether the panel is going to be curved or not.
 2. Compute the coordinates of the three vertices if the panel is flat, or two vertices plus arc length in case it is curved. The main difference with curved panels is the way the coordinates of the points are computed.
 3. Create a *FlatRecgeo* or a *CurvedRecgeo* object based on the radius value. From now on it will be referred to as “LeftPanel”.
 4. Add nodes independently with the corresponding method of “LeftPanel”.
 5. Retrieve and store the nodes (ID and coordinates) that belong to edge 2 of that object, as it will be necessary to add a new sub-mesh later on.
 6. Create the elements with the corresponding method of “LeftPanel”.
 7. Store nodes and elements in global variables, where every node and element of the final panel will be kept. Information about the entity to which elements belong (skin) and about nodes which are part of a side of the final panel are stored as well to ease the application of boundary conditions or loads on them, and the assignment of material properties to elements once the mesh is completed.
 8. Return nodes belonging to edge 2 so they can be used to assemble next sub-mesh.
- *CreateRightPanel* method: since geometry is built from left to right, this is the last method that must be called. It creates the skin from last stringer to side 2 of the panel. The procedure is basically the same. The main differences lie in steps two and four: as this sub-mesh is going to be added to the rest of panel mesh, vertices coordinates are read from the edge 2 of last sub-mesh created, which can belong to last stringer, or to left skin if there are no stiffeners. Moreover, nodes are added with the corresponding “RightPanel” method that manages the fact of having the new edge 4 in common. Finally, steps five and eight are not necessary as this is the last sub-mesh that will be created.
- *CreateIntPanel* method: in charge of creating intermediate skin sub-mesh, which is the piece of skin in between two consecutive stringers. This method is called just if the number of stringers is greater than two. The procedure is exactly the same as to create the “RightPanel” object. The only difference is the width of this skin mesh, and the fact

that ID and coordinates of the nodes belonging to edge 2 must be stored and returned for next geometry.

The following seven methods are responsible for the mesh generation of each type of stringer profile. Each one follows a specific structure to build the geometry based on its profile shape. As a result, they have important differences due to geometry but the methodology is quite similar. “Z” profile will be explained in depth as an example. The structure is the following:

1. The profile must be divided in smaller sub-meshes. This division depends completely on the geometry of the profile. It is important to take into consideration the order in which these sub-meshes will be joined afterwards, in order to store the minimum data possible. Since information of nodes in a common edge is required in order to add the next sub-mesh, this information must be completely used before storing another edge. This will be understood better in a moment.

In this case, creation of the profile is carried out as follows. The profile is divided into three sub-meshes: bottom flange, web and top flange. These sub-meshes are implemented in that order: starting with the bottom flange which is joined to the previous skin sub-mesh, then the web and finally the top flange.

2. Check whether it is a curved or a flat panel. This is crucial as it affects not only the bottom flange (which can be curved or flat) but also how coordinates and geometry is computed. In flat panels, the “Z” stringer web is vertical and both flanges horizontal, whereas in a curved panel bottom flange is curved following the arc of the skin, web is orthogonal to this curvature, and top flange is flat and orthogonal to the web. All these geometrical adjustments are carried out mainly by the aid of the rotation method and cylindrical CSYS explained in *FlatRecGeo.cs* and *CurvedRecGeo.cs* classes.
3. Bottom flange object is created following the same procedure as in skin sub-meshes: vertices are defined, the object is instantiated, nodes are created and joint to the previous sub-mesh through edge 4, elements are added, edge 2 nodes are then stored for next sub-meshes, nodes and elements are added to global variables, and information about element entities or nodes located on the sides is collected for later use.
4. Then Web object is created following the procedure just explained above. The only difference is that the new edge 2 (which is necessary to assemble top flange sub-mesh) cannot be stored in the same variable where the edge 2 of the bottom flange is stored, otherwise the next skin sub-mesh will not be able to be added to this stringer through the correct edge. Then, the second edge of the web is stored in another variable, and it is used to create the Top flange object in next step.
5. Top flange object is created and joint to the geometry through Web’s second edge. It is not necessary to store Top flange’s edge 2 as no more sub-meshes will be assembled there. Now, the whole profile is created, and in order to

keep adding sub-meshes to the existent mesh, bottom flange edge 2 that was stored at the beginning must be returned for later use.

This procedure allows creating the geometry of almost any profile. For profiles “I” and “Omega”, some of the sub-meshes are created within two common edges. Apart from that, the way of implementing the mesh is quite repetitive. The following image shows the sequence order in which each sub-mesh of a stringer is created:

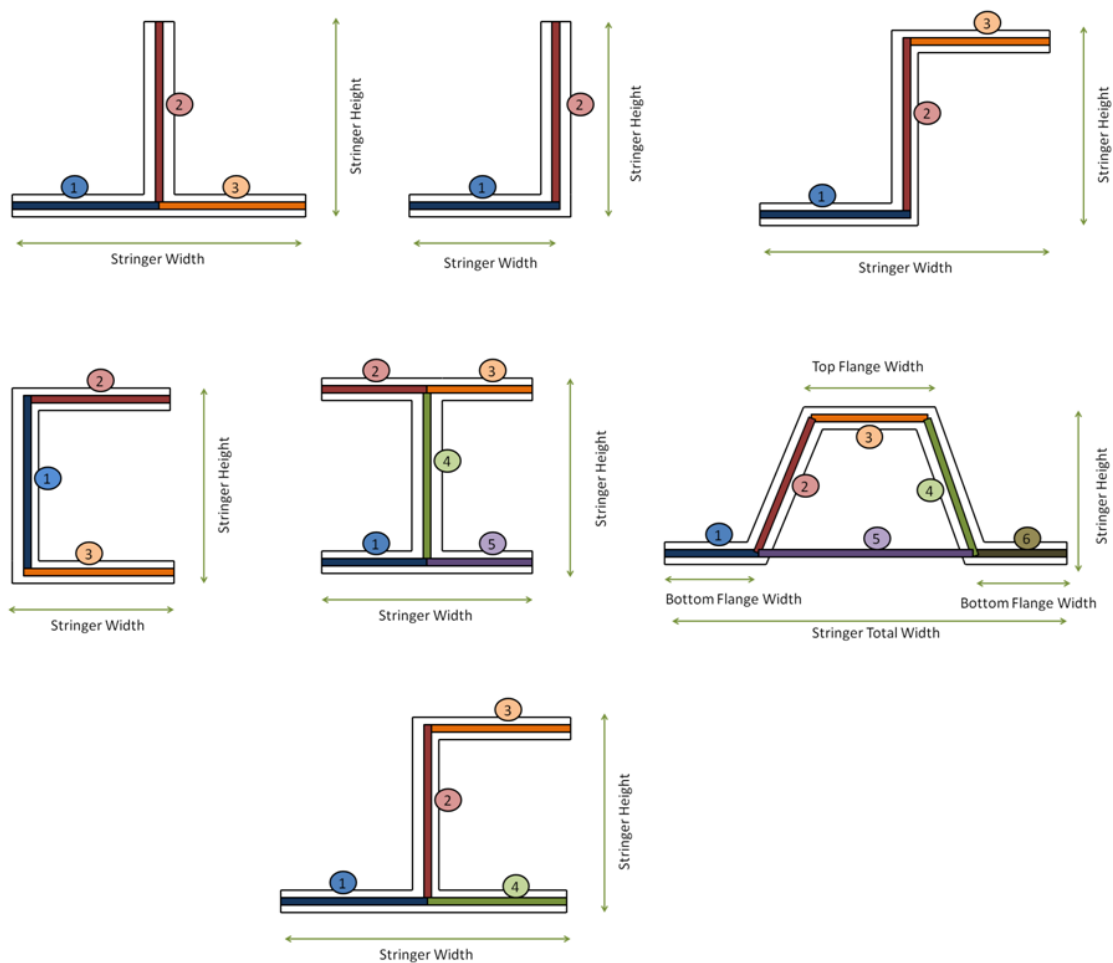


Figure 5.15: Schematic representation of the sequence followed to create the geometry and mesh of the main seven profiles of the application.

- *CreateWholePanel* method is the most important since it actually creates the complete panel mesh. It starts checking that the width of the lateral skins F_x has been computed correctly. If so, it creates the panel in the following order:
 1. First, the left lateral skin is created.
 2. Then, if the number of stringers is greater than zero, it creates the first stringer. Since the stringer profile was one of the arguments of this class, it already knows which profile needs to be created.
 3. After that, a loop starts for the rest of stringers, provided that there are at least two. This loop will create an intermediate skin and consecutively another stringer. This loop will be repeated till all the stringers have been created.
 4. Finally, the right lateral skin is added to the model, completing the mesh.

If F_x was not computed correctly, none of the previous code is actually executed and the method does not create any panel, returning an error.

- *PrintMSHData* method stores some text in a string to be later written in a *.msh* file, which can be opened with gmsh [21] to visualize the mesh of the stiffened panel. It has no arguments since all the information is already available. But in order to call this method, firstly the previous method must be called to create the panel. The structure of the text contained in the *.msh* file can be found in *Appendix A*.

5.4. The Load and Boundary Conditions manager tool

The Load and Boundary Conditions (BC) manager tool receives the applied loads and BC values introduced by the user and adjusts them to be understood by the solver. Regarding the loads, this process is mainly the translation from the total load per unit length applied on each side of the panel to actually the individual load value applied to each node belonging to that side. This process depends on the type of element and the element width. Since the second order quadrangular element is the one that will be used within CADEC environment, that is the case that will be explained into depth. Basically the load distribution is such that nodes on the corners of the element receive one sixth of the total load applied on that element, whereas the central node of the edge receives four sixths (see figure 5.19).

In regards of the Boundary Conditions, the tool needs to translate conceptual language (*simply supported, clamped, free or symmetry*) into the actual Degrees Of Freedom (DOF) restrained for each node. Additionally, depending on the input loads, some extra DOFs may be constrained to avoid panel oscillation within its plane and ensure solution convergence.

The above mentioned tasks are accomplished by the following class: *INPUTdata.cs*.

5.4.1. Handling loads and boundary conditions: INPUTdata.cs class

A. Aim and arguments

Aim and arguments:

The following class manages all the necessary information regarding boundary conditions and applied loads of our panel. This class in combination with all the panel mesh data already created allows the subsequent creation of an INPUT file. This file can be submitted to analysis by other software such as Abaqus•, or more importantly, the later call to BMI3 solver to directly obtain the first buckling modes of the structure.

It must be called after a mesh has been created, as it requires elements information to assign material properties, and nodes information in order to apply boundary conditions and loads correctly.

Then, arguments required to create an instance of this class are: a pointer to the SPmshBuilder object representing the panel mesh, type of BC applied to each side of the panel and application mode (which can be just to the skin or both to skin and stringers), and values of shear and/or normal lineal loading applied on each side.

In order to understand the following methods it is relevant to define the different Degrees Of Freedom (DOFs) that the solver BMI3 will use. As any rigid body in the space, each node has six DOFs, three of them defining translation and the other three to rotation. Hence, DOFs 1, 2 and 3 correspond to translations along X, Y and Z axes, respectively. The rotational DOFs are not so intuitive. They are defined in accordance with the classical theory of plates, so that DOFs 4, 5 and 6 are respectively θ_x , θ_y and θ_z . It must be noticed that θ_x does not coincide with rotation about first DOF (θ_1 according to the next figure) and neither does θ_y with θ_2 . On the other hand, θ_x defines the rotation in XZ plane whereas θ_y does it in YZ plane.

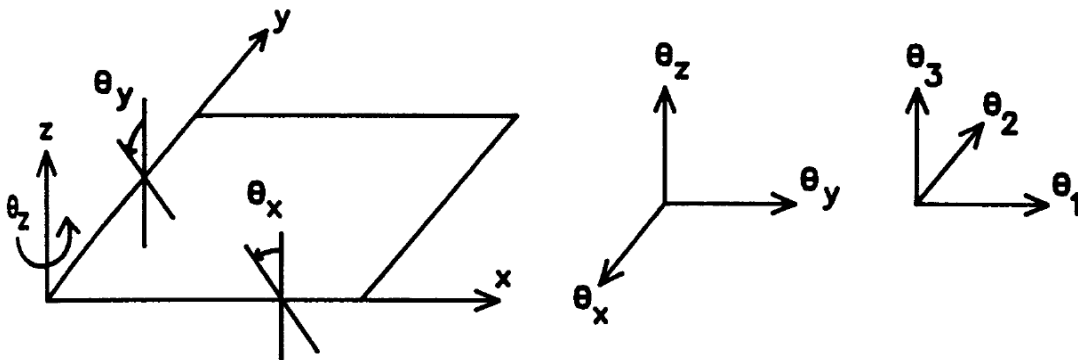


Figure 5.16: Definition of the rotational degrees of freedom used by the tool. Source: Figure 5.2 of [6].

B. Methods

There are three methods in charge of presenting the information of the problem in a way that BMI3 or Abaqus® can solve it. The first two store in a string the text content of the *.inp* file, so such file can be created later on. The third computes some data required by the former methods.

However, it is when creating an instance of that object, that all the important calculations and assignments are performed. The process followed is:

1. Assign the value of the arguments to the local variables of the class, as properties. Depending on the type of BC applied, the number of DOFs that must be constrained varies. Therefore, at this point the size of the variables containing BC information is set. Furthermore, depending on which axis is parallel to the extrude direction of the panel, the corresponding DOF are updated. If the panel has been rotated so that the X axis is parallel to extrude direction, DOF 1 and 2 are switched, in addition to other adjustments to ensure the proper functionality of the program under the new CSYS.
2. Boundary Condition application: It will go through all four sides of the panel, and for each side it will perform the following operations:

2.1 First, the ID of the node that is located in the middle of that side must be retrieved. This is necessary in order to apply additional boundary conditions to the panel in case normal loads and no symmetry are applied. Even though applying two compressive loads in parallel sides of the same value leads to a mechanic equilibrium state, from the solver point of view the panel is still capable of vibrating on its plane: XY plane. Then, an additional DOF must be restrained on the two nodes in the middle of the other two sides, creating a symmetry effect with respect to the mid fiber of the panel, and then enabling the solver to converge to the solution (see figure 5.17).

2.2 Second, it browses all the nodes that belong to that side of the panel, and based on the BC applied, the corresponding DOFs are constrained. Should BC be applied on the stringers as well, they are added at this point for sides 1 or 3. The following table summarizes DOF that are restrained for each type of BC. Please notice that they are **referred to the old CSYS shown in Figure 5.1**, which does not correspond to the global CSYS of CADEC. Therefore, in Figure 5.16, positive θ_x is defined on side 1, and positive θ_y on side 4.

BC	Sides	DEGREES OF FREEDOM CONSTRAINED					
		1 (u_x)	2 (u_y)	3 (u_z)	4 (θ_x)	5 (θ_y)	6 (θ_z)
SS	Sides 1 and 3	-	-	0	0	-	-
	Sides 2 and 4	-	-	0	-	0	-
CC	For every side	-	-	0	0	0	-
SYM	Sides 1 and 3	-	0	-	-	0	-
	Sides 2 and 4	0	-	-	0	-	-
FREE	For every side	-	-	-	-	-	-

Table 5.1: Degrees of freedom restrained for each boundary condition type depending on the side.

2.3 Finally, it is checked whether additional BC must be applied. This depends on the combination of loads and BC applied. The different cases are summarized in the following table.

Load	Condition	Constrained DOFs	BC application
Normal Loading	Sides 1 and 3 do not have symmetry BC	2 (u_y) = 0	Nodes in middle of sides 2 and 4
	Sides 2 and 4 do not have symmetry BC	1 (u_x) = 0	Nodes in middle of sides 1 and 3
Shear Loading	Loads $N_s > 0$	1 (u_x) = 0	Node corner sides 1 and 2
		2 (u_y) = 0	Node corner sides 3 and 4
	Loads $N_s < 0$	1 (u_x) = 0	Node corner sides 1 and 4
		2 (u_y) = 0	Node corner sides 2 and 3

Table 5.2: Additional degrees of freedom restrained depending on the type of applied load to ensure that the panel has no rigid body motion.

In case the panel is subject to compressive loads and some of the sides are not symmetric, it is necessary to constrain the possible oscillations of the panel on the XY plane. If none of the sides 1 and 3 are symmetric, despite being in mechanical equilibrium due to the loads applied, the panel could move in DOF 2: "Y" direction. Therefore, this DOF must be constrained, and the nodes selected to apply such additional boundary condition are located at the middle of sides 2 and 4 (on a symmetric axis) so that results are not affected since this two nodes should not move on that direction. Same reasoning applies for the second case.

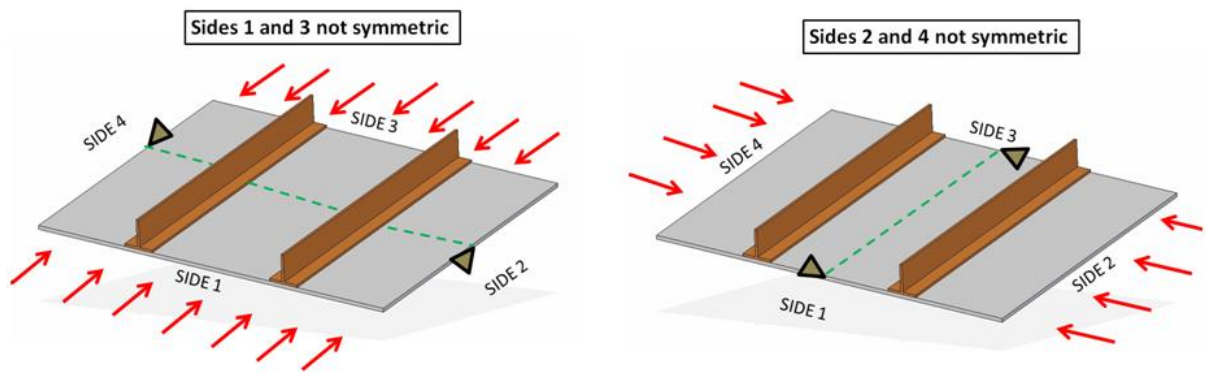


Figure 5.17: Schematic representation of additional constraints required to ensure problem solution of panels subject to in-plane compression.

On the other hand, if the panel is subject to shear loads it is necessary to constrain the possible rotation of the panel about “Z” axis: DOF 6. This occurs since under shear loading, despite being the module of the loads the same so that forces were in equilibrium, moment M_z will never be cancelled. In this situation it is necessary to constrain that DOF for the panel. There are several possibilities, but the most commonly used is to constrain the motion of two of the corners so that the panel cannot rotate. These nodes will have constrained DOFs 1 and 2 as shown in the figure. Ideally, it is not necessary to constrain both 1 and 2 DOFs in one of the nodes, but it was decided to apply both for convergence requirements of our solver.

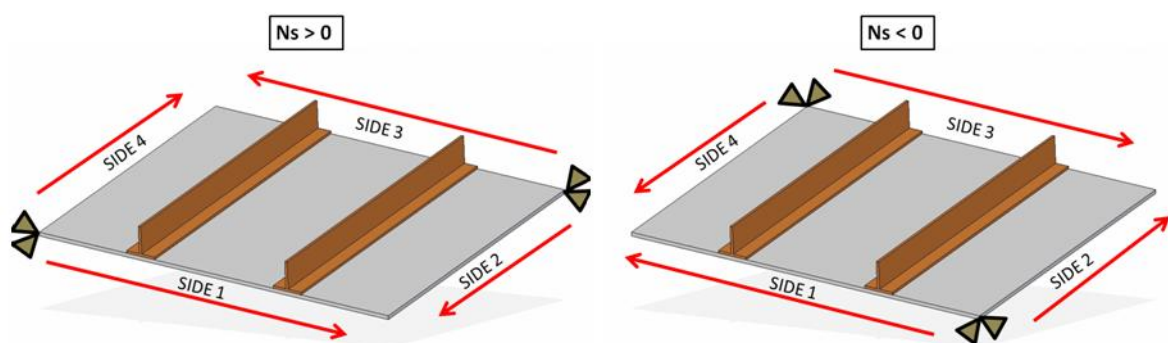


Figure 5.18: Schematic representation of additional constraints required to ensure problem solution of panels subject to in-plane shear loading.

Finally, in the specific case of non-stiffened plates, the drilling rotation (DOF 6) is constrained for all nodes. This kinematics is similar (but not the same) as that of element S9R5 in Abaqus®.

The information stored concerning BC is the following: node ID, DOF that is going to be set, and the value of that displacement or rotation. As displacements or rotations are restrained, the value is set always to zero. But it could be possible to impose a specific displacement in order to obtain the required load value that causes it.

3. Load application: the values introduced by the user are lineal forces, so the values result from the ratio between the total forces applied and the panel width of the side where they are applied. However, our solver needs to know the equivalent force absorbed by each node. Depending on the type of element, the load share value (relative load applied on a specific node) varies. For triangular elements and first order quadrangular elements, the total force absorbed by that element is split equally into the two nodes on the corners. However, for second order quadrangular elements, the load share is different: one sixth for each of the nodes on that corner, and two thirds for the node in the middle. Then, at this step it goes through all elements in the panel performing the following operations:
 - 3.1 First, check whether that element contains some nodes on any of the sides of the panel, and then potentially subject to the application of forces.
 - 3.2 Second, the length of the element is computed in order to know which fraction of the total force is absorbed by that element. The calculation is different for curved and flat elements.
 - 3.3 Third, the distribution of the load into each node is computed. This value is dependent on the side, due to the fact that a node on a corner of the panel could have a different load share as it belongs to side "A" from the load share as it belongs to side "B". That is independent on the applied load value, but simply the element width on one direction could be different from its width in the other. The value results from multiplying the specific ratio that depends on the element (one half, one sixth or two thirds) times the element length. The distribution factor dependent on the element type is shown in next figure.

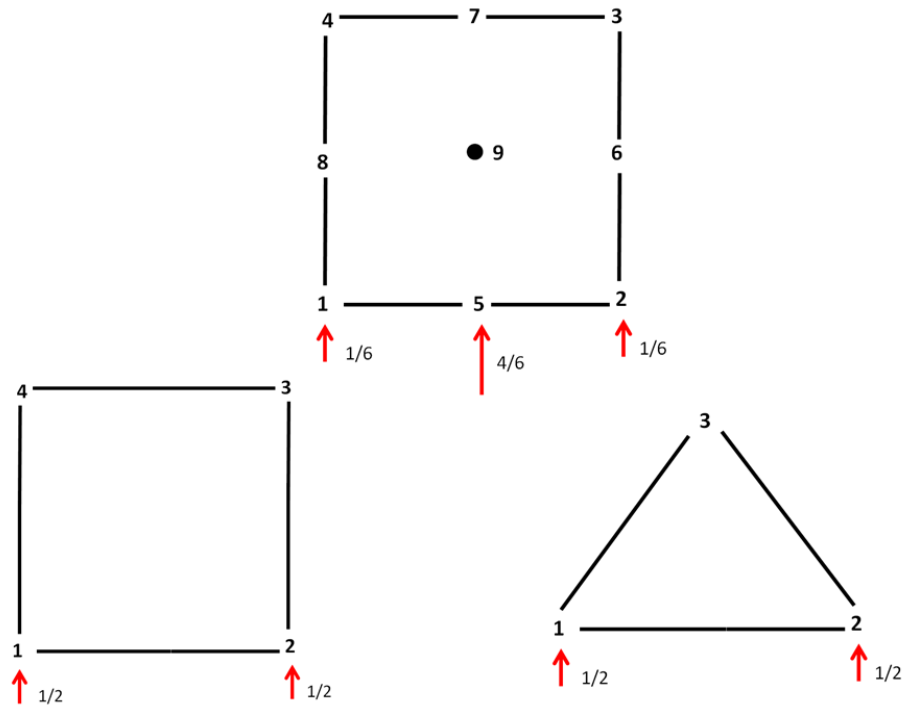


Figure 5.19: Load share for each type of element.

Once the load share has been computed, it is time to obtain the actual value of nodal forces. The parameters that must be stored again are: node ID, DOF in which load is applied, and the nodal force value in that DOF.

To size the array containing all load information it is necessary to check the applied loads and whether they are applied on the skin or both skin and stringers.

Finally, the last parameter (nodal force) is computed by multiplying the load share of each node times the lineal force applied on that side. As the load share has length dimensions, and the input load is lineal force, result has force units.

- *PrintINPData* method stores in a string all the information necessary to create an input file. The format depends strongly on the targeted software that will use it. This method generates the *.inp* data required by BMI3. It is important to notice that in order to run the problem material information is also required. However, BMI3 (the executable version of our solver) reads that information from a *.dat* file, which is generated independently in the final application based on the material information stored in the database. A description of both files can be found in *Appendices B* and *C*.
- *PrintAbaqusINPData* method stores in a string the information required to create an input file. In this case, it is generated following Abaqus® script procedure. Abaqus® requires material information included in the *.inp* file, so the matrices A, B, D and H are sent as arguments to the method as well as the number of materials. For that reason,






as opposed to BMI3, Abaqus® just requires one file to solve the problem. A description of the file can be found in *Appendix D*.

- *GetOrientationsForAbaqus* method calculates the components of material directions 1 and 2 in order to assign material directions to a curved panel of composite or omega profile in flat panels, necessary for .INP file for Abaqus®.

5.5. The solver: BMI3

The solver is BMI3 [6]. It uses a 9-node isoparametric, fully integrated element with relaxation of the Kirchhoff condition [33, 34]. Relaxation is necessary in order to avoid locking, since fully integrated elements result in excessively stiffened elements. Full integration is used to avoid zero-energy modes, which appear if using reduced integration [5, 27].

After the implementation of full integration, it is possible to analyze plates with high aspect ratios (thickness to in-plane dimensions ratio) without any revelation of zero-energy modes. An example follows of a plate with aspect ratio of 5 in order to illustrate the robustness of the fully integrated elements.

Panel Information	
Name :	SS plate sym uniaxial loa
Buckling Results	
First Eigenvalue	1176.976
Second Eigenvalue	3231.783
Third Eigenvalue	3968.62
Fourth Eigenvalue	4257.198
Fifth Eigenvalue	4328.622
Relative displacements	
	0.32
	0.24
	0.16
	0.08
	0

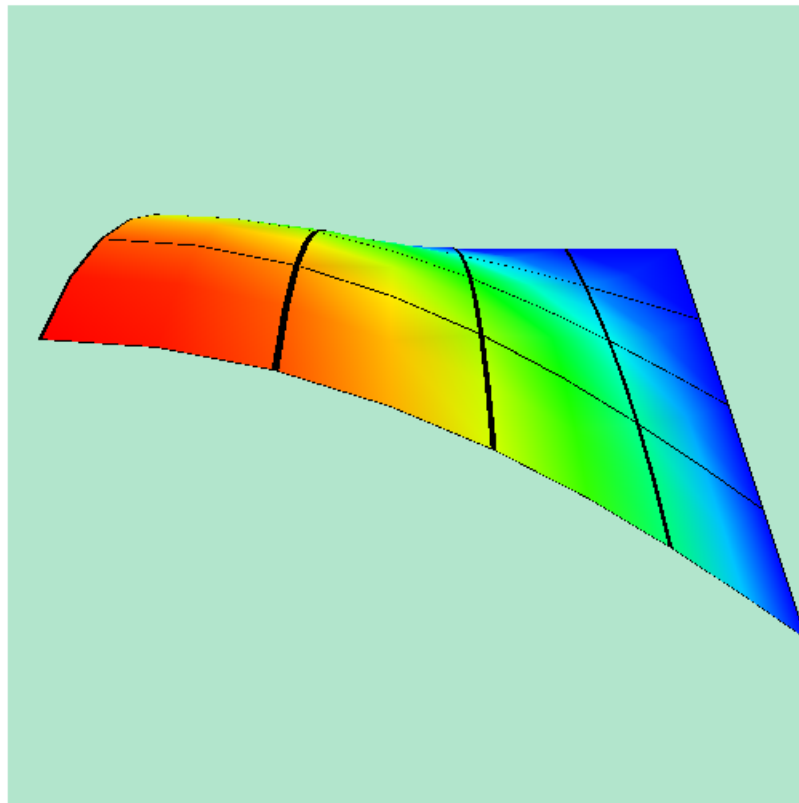


Figure 5.20: CADEC results for 9-node fully integrated/relaxed element, with no zero-energy mode.

Apart from that modification in the solver code, the only novelty is the addition of a new class in order to enable communication between the rest of the application (built in C#) and the solver (written in Fortran).

Despite the fact that all the code necessary to solve panels subject to shear loads was implemented, the current solver is not capable of solving this type of problems. The reason behind that behavior follows.

In structural buckling, all eigenvalues are real values. For axial loading, all the eigenvalues are also positive. The eigenvalue of most interest is the smallest since it gives the lowest load at which the structure will buckle. The solver used by CADEC [2] is optimized for speed by computing only a few of the smallest eigenvalues, assuming they are all positive.

However, for shear loading the eigenvalues come in positive/negative pairs because the response of an orthotropic material, plate or stiffened panel is indifferent to the sign of shear. Thus, the solver tries to find the smallest, which in this case is the largest negative value and it crashes. All the remaining solvers available in the public domain that can compute the smallest positive value for this problem are more than 100% slower than the optimized BMI3 [6], which prevents their use in a web application as CADEC [2]. Since performing an important modification on the code of the solver is out of the scope of this project, it is not currently possible to solve this type of problems, but it should in a future version.

5.6. The visualization tool

The visualization will be carried out in the Web, and will be further discussed into detail in another chapter. However, the application itself contains an extra class in charge of creating a POS file so that the deformed shape of the panel can, in addition, be shown by gmsh [21].

This task is accomplished by the following class: *POSdata.cs*.

Furthermore, it is also possible to show the mesh itself before actually running the problem. In order to visualize the mesh of the stiffened panel with gmsh, it is necessary to download a file with the text provided by the method *PrintMSHData*, which can be found in the class *SPMshBuilder.cs*.

5.6.1. Results post-processing using gmsh: *POSdata.cs* class

A. Aim and arguments

This class stores in a string all the information in order to visualize the deformed shape of the panel in the free software gmsh [21]. The text of this string can be written in a file and the user should be able to download it in order to visualize the results offline. Current version just supports second order quadrangular element type for the mesh.

The only arguments required to create an instance of this class are a pointer to the `SPmshBuilder` object (the panel mesh) and the array containing the displacements from the solver.

B. Methods

PrintPOSData is the only method, and as its name suggests it prints the node initial coordinates and the new displacements in a string. If the text contained in this string is written in a *.pos* file and opened with `gmsh`, the deformed shape of the panel will be shown. It also defines some default properties for `gmsh` viewport such as the displacements scaling factor. Since the final web application will contain its own visualization tool, this method is not currently available for the user.

Chapter 6

6. Web Environment

6.1. CADEC

At this point a set of tools for the analysis of composite stiffened panels has been developed. However, the application lacks a user interface. In other words, tools are available but there are no gloves to handle them. There are several possible approaches to this problem. One is to implement the tool in a Windows Form, in which the user introduces data in textboxes and submits information using buttons. Another option would be compilation of an executable, which would read the information from a file, process it, and return another file with the solution requested. Both solutions are feasible but dependent on the operating system of the user's computer, limiting distribution. The second option has an even worse drawback; it may be very tedious, error-prone and time-consuming to require the user to introduce the data from a text file. A combination of both alternatives is the application SPD. Therefore, as explained in the introduction of this project, this set of tools will be implemented as a web application. Furthermore, in order to make this tool even more powerful, the tool will be combined with an existing website providing further features to the user. The interface will be created by the implementation of the tools within CADEC's environment. CADEC (Computer Aided Design Environment for Composites) [2] is a web application developed by Professor E.J. Barbero and hosted by the Department of Mechanical and Aerospace Engineering of West

Virginia University. CADEC performs composite materials analysis, and shares a strong relationship with the purpose of our application.

Implementing the tool inside an existing website has benefits:

- The website already has a structure, eliminating planning regarding page layout, style or organization. A database in the server and master page already exist. Code can be reused in general.
- Existing functionality adds value to the tool. For instance, several pages of CADEC already help the user to create matrices, fibers, and laminates. Instead of duplicating those functionalities, the tool can simply handle the communication between these objects. In addition, CADEC also provides user authentication.

On the other hand, this approach involves additional difficulties including:

- Freedom is limited within an existing structure already exists, and new pages must share the appearance of other pages. This may conflict with some predefined expectations for the web application.
- Most of the high-level features encountered in CADEC improve the overall functionality of the page but require additional code and complexity. Most of these issues will be highlighted during the explanation of the implementation process.
- Additional time is required to format the structure of the page to existing procedures, despite the robustness of the existing website. This factor is interrelated with the complexity concern.

Overall, the possible benefits of the tool provided by CADEC's environment clearly outweigh the additional difficulty that may arise.

Before moving on to the implementation process, it is important to understand CADEC's structure and functionality. There are two possible views of the application: the user's view and the programmer's view. The former provides enough background to understand how to use this powerful tool, including the new module for analyzing stiffened panels. The latter is relevant for discussion specific to this project, and a brief overview will accompany the later explanation of the implementation process.

6.2. Databases and stored procedures

Though CADEC already contains a database (DB) [31] to store all objects the user creates, some modifications are required to allow the storage of stiffened panel objects. A database is simply a structured collection of data. The language used to communicate with this DB is SQL (Structured Query Language). Specific examples of SQL as used to create tables or to edit records can be found in Appendix E.

First a new table must be created in order to store the stiffened panel objects. A table consists of rows and columns: each row represents a different object, and the columns define the properties of each object. A table contains different records or rows, and each record is defined by a set of fields, each being a single piece of information, structured in columns. Then, in order to create a table it is necessary to state the minimum amount of fields required to completely define a stiffened panel object. Each of the columns representing a property requires a name and a data type. To quickly create the same table in different DBs, a SQL script is used.

After creating the table, it is possible to access the data using SQL commands. However, in order to keep code as simple and clean as possible, the SQL code necessary to manage the DB will be stored in files called stored procedures. A stored procedure is a subroutine available to applications that access a relational database system. They consolidate and centralize logic that was originally implemented in applications. In our case six different stored procedures will be defined, each of them dealing with one of the following actions:

- Creation of a new record (insert a new panel in the DB).
- Deletion of an existent record (remove an existent panel from the DB).
- Edition of some fields of a record (update some properties of the panel in the DB).
- Retrieval of a single field of a record from the DB.
- Retrieval of a list of all the records (list all the panels that are in the DB).
- Retrieval of a complete record of the DB, including all the different fields for that record.

To centralize the code and avoid code-repetition these subroutines are stored within the DB, making the application very robust.

Chapter 7

7. Implementation

7.1. User's view

First of all, CADEC can be accessed following the URL found in [2]. The main page is shown in the following image. Let us explain what can be found in this page.

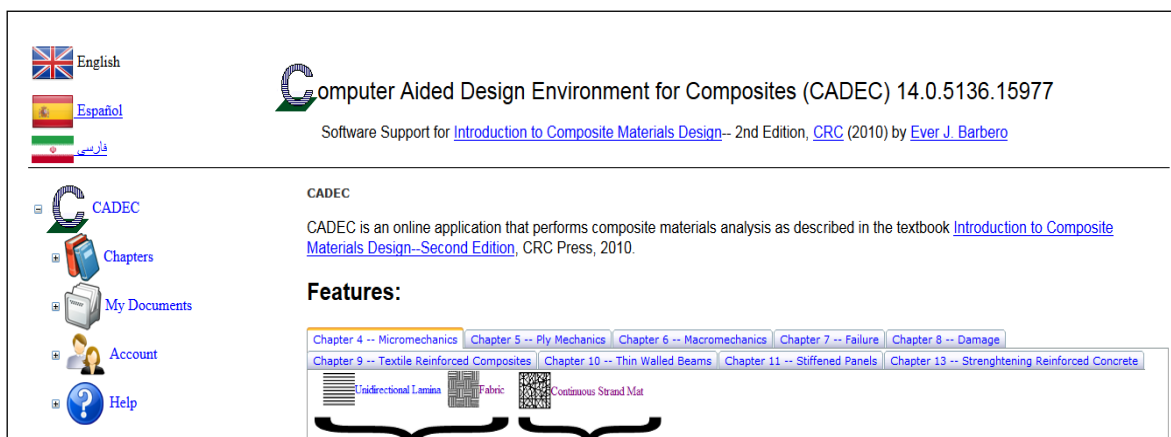


Figure 7.1: Main page of the web application CADEC.

In the upper-left corner of the page, three elements allow to change the language of the website. Current languages available are English, Spanish and Persian. Just below them, also in the left side of the screen, the main navigator tree can be found. There are four main sections: Chapters, My Documents, Account and Help. Let us proceed to a brief review of each of them:

- Chapters' section: It is relevant to notice that CADEC is structured following an analogy to the textbook [7]. Thus, it is possible to find different kind of composite materials analyses such as damage, micromechanics, macromechanics, etc. within this section. These tools are organized in chapters as they appear in the previously mentioned book. In other words, chapters' section contains different pages that are in charge of carrying out computations. Every page that performs any analysis based on given input data can be found here.
- My Documents' section. It contains all the pages where the user provides input data to create objects, objects that may be used later in Chapters for calculation. Examples of objects could be fibers, matrices, laminates or stiffened panels. These objects created in My Documents are stored in a database in such a way that can be accessible from Chapters and also they will remain in the server for future sessions. In contrast, the results provided in Chapters' section are deleted once the user leaves the current page.
- Account's section. As it can be inferred this is the place where the user can manage its account properties such as username, password, email address, etc. It should be noticed that an account is not mandatory to access CADEC. However, it is necessary to be able to actually create objects in My Documents or carry out the analysis in Chapters. As guest, the user would be capable of reading information about what each page does but will not be able to access directly the pages inside My Documents and My Chapters. Nevertheless, registration is free as long as the email provided is related to educational institutions (universities, schools, etc).
- Finally, there is another section named Help, which includes a complete user's manual, forum and FAQs regarding CADEC. This section can be fully accessed as a guest.

Among the previous sections, our application will have an impact just in the two first. Chapters will include the pages where the buckling analysis of the stiffened panel is performed. My Documents, on the other hand, will be contain the pages where the user can see the panels already stored in the database (DB), edit them or create new panels. So far, it is quite clear that CADEC's use is very intuitive.

Finally, the actual content of each page can be found on the right of the navigator tree, so that the navigation tree and the language selection are always available in the screen.

7.2. Programmer's view

This section will provide an overview of CADEC's code organization. As it is shown in the following diagram, CADEC is formed by six main projects. Each of the projects contains specific type of information, so the organization of the website is maintained.

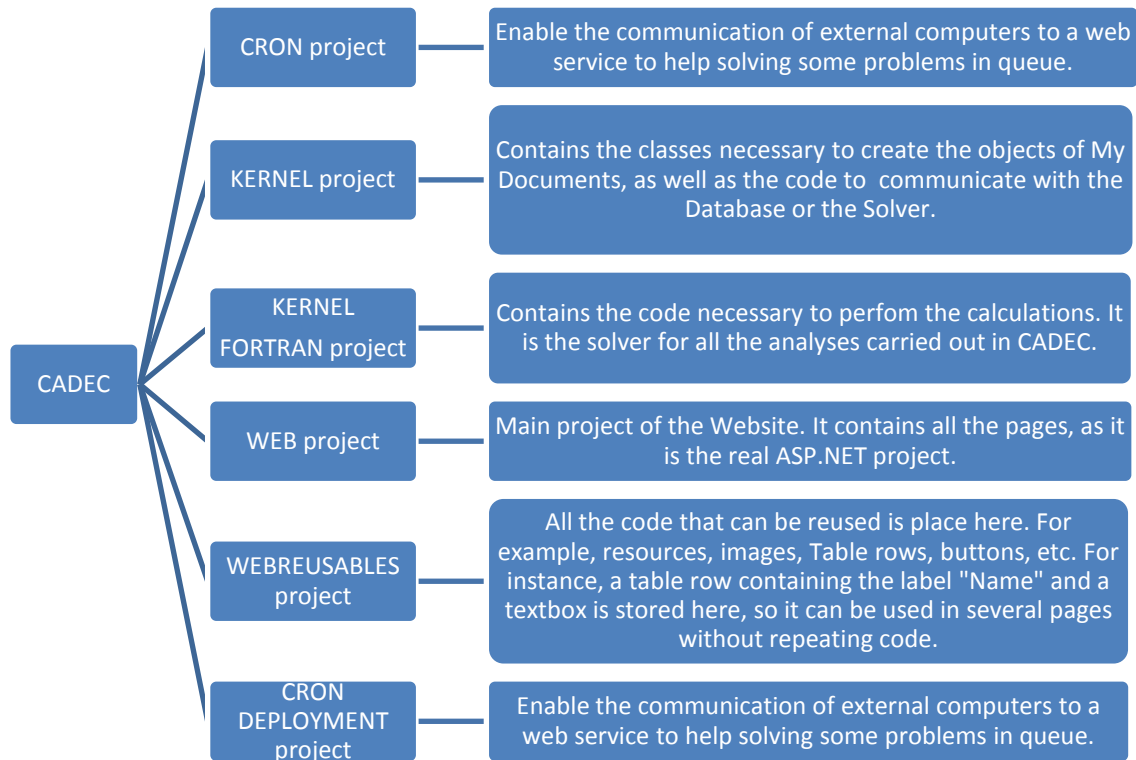


Figure 7.2: Scheme of the different projects forming CADEC.

In regards of our application, four of the projects will need to be modified: *Kernel* project, *KernelFortran* project, *Web* project and *WebReusables* project. By explaining how our application will be implemented in each of the projects, the main purpose of each project will be revealed.

7.2.1. Kernel Project: definition of CADEC's objects

KERNEL is a C# project formed by several classes that contain the code necessary to define objects, to communicate with the database and to call the solver. It is divided into different folders, each of which contains the code for each kind of object. Therefore, main object types that can be found here include: environments, fibers, matrices, laminas, loads, materials, panels, sections, strains, etc. It contains a folder named Utilities, and here is where the class that enables communication between the solver (in Fortran code) and the rest of C#

code will be placed. It plays the role of translator between the two different languages, and without that class it would not be possible to access Fortran functions located in *KernelFortran*.

What is our contribution to this project? For starters, this is the place all the previous tools must be placed. As it is shown in the following image, the new folder named *StiffenedPanels* will contain the mesh generator tool (stored in *SPmesher* folder) and also the tools managing Load, BC and post-processing, which will be stored in a separate folder named *SPDtool*. All these six classes have been discussed exhaustively in previous chapters, and their code has not been modified. However, there is a new class that must be created. This class is called *StiffenedPanel.cs*.

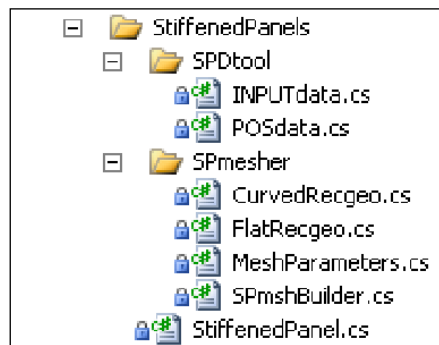


Figure 7.3: Files added to Kernel Project for the web implementation of our application in CADEC.

The *StiffenedPanel.cs* class creates stiffened panel objects. What does this class look like? Firstly, it contains the definition of its main properties. Its properties include:

- Geometric properties: panel width, panel length, panel radius of curvature, panel pitch, offset of the stringers (positive value coincides with positive Y axis direction), number of stringers, type of stringers, stringer height, stringer total width, top flange width and bottom flange width.
- Material properties: skin's material ID, Bottom flange's material ID, Web's material ID and top flange's material ID.
- Load and BC properties: load and BC application type (skin or skin and stringers), and values for each side.
- Extra properties: These extra properties are not stored in the database, but are needed for some of the pages. Such properties are Curved panel (value of true if it is curved and false if it is not), stiffened property (true if it contains stringers and false if it is just a plate) and buckling load type property, which determines whether the buckling analysis will be carried out under normal loading or shear loading.
- Global variables for BMI3: Here are defined the variables required by our solver in order to solve the buckling problem. Such variables include information about the mesh (number of nodes and their coordinates, number of elements and their connectivity), about the loading and BC (similar to the properties already defined regarding these issues, but in a specific arrangement for BMI3), about the materials (in this case BMI3 needs the ABDH matrix components for each type of material involved in the problem, as well as the information of the material

applied on each element), internal parameters for the solver (such as number of roots (eigenvalues) required or the subspace dimension, and finally the output variables which are the eigenvalues and displacements of the first mode. All these variables are not stored either on the database. In fact, they are initialized just before calling the solver for calculations.

After all the properties have been defined, some constructor methods are defined in order to instantiate a new stiffened panel's object. The three constructors are:

- *StiffenedPanel(Id)*: This constructor just requires the ID of the stiffened panel in the database (*Id* variable) in order to create a new object. The code inside this method basically uses the stiffened panel ID and the user's ID to communicate with the Database and retrieve all the properties stored there. It is used within the method *All(owner)* that will be explained later, that is the reason why the method does not require explicitly the user's ID (*owner* variable).
- *StiffenedPanel(Id, owner)*: It requires both the ID of the stiffened panel in the database and the user's ID in order to create a new object. The code inside this method basically uses the stiffened panel ID and the user's ID to communicate with the Database and retrieve all the properties stored there. In order not to include all the code required to communicate with the database, it uses a stored procedure, which means that the missing code is stored in the database itself and is executed from there.
- *StiffenedPanel()*: This constructor is used to create an instance of a stiffened panel that has some default values for the properties. This will be used to provide the default values of properties in My Documents when a new panel wants to be created.

Afterwards, other methods are defined still to perform actions with the database. These methods allow to create a new panel in the database, modify some of the values of a panel that already exists in the database, delete panels from it, etc. It should be explained that all these methods use stored procedures in order to perform the previous actions. The main methods are described next:

- *All(Owner)*: As it was previously advanced, this method uses one of the constructors in order to return a list with all the panels that the user has stored in the database, along with all the properties of each panel. This method is used in *MyDocuments/StiffenedPanels/Default* page, where a table shows all the panels of the given user with summarized properties so that the user can decide whether any of the panel needs to be deleted or edited. It uses the procedure "*CADEC_StiffenedPanels_All*" in order to retrieve the data.
- *AllStripped(Owner)*: Similar to the previous method, this one returns a dictionary containing the ID of each panel and its name. This is used for all the drop-down Lists that are used in the pages. For example, when deciding which panel is going to be submitted to analysis, there is a list that includes the names of the panels the user has in the database. It is not necessary to know all the properties of all of them, just the name. Then, after a panel selection has been made, the properties

of that specific panel are retrieved. It uses the stored procedure "*CADEC_StiffenedPanels_All_Stripped*".

- *Delete(Id,Owner, DeleteDependencies)*: This method is in charge of deleting a panel from the database. In order to achieve that, the method requires the panel's ID and the user's ID of the targeted panel, as well as additional information regarding whether dependent objects must be deleted as well. Since Stiffened Panel is an upper level object, that will not be the case since there is no object in a superior hierarchy that needs to be deleted. However, there may be in the future, so the option is available. In order to understand this issue let's consider a laminate. The user may have used that laminate as the skin's material of a stiffened panel. Then, if at some point the user decides to eliminate that laminate, the stiffened panel will still have the ID of that material as the one used for the skin. This could lead to errors with that stiffed panel in the future, since it is pointing to a non-existent object. For that reason, in that case it may be necessary to delete the stiffened panel as well. That is the purpose of that last variable: decide whether dependent entities must be deleted as well or not. It uses the stored procedure "*CADEC_StiffenedPanels_Delete*".
- *Insert(...)*: As its name implies, this method inserts (creates) a new panel object in the database. All the arguments have been removed for simplicity, but basically it requires the values of all the properties that will be stored in the database and that define a specific panel. These properties were explained at the beginning of this section, properties include geometry, materials, load and BC conditions. They also include the user's ID but since the panel does not exist yet, no panel ID is required. Instead, it generates a new ID for the panel when adding the records to the table. It uses the stored procedure "*CADEC_StiffenedPanels_Insert*".
- *Update(...)*: Very similar to the previous method. Instead of creating a new panel, it updates any modification of the parameters of the panel. Then, it accesses the record of the panel with the ID given and overwrites the values. It uses the stored procedure "*CADEC_StiffenedPanels_Update*".

Finally, there are three more methods contained in this class. At this moment all the methods regarding object instantiation and database communication have been explained. However, it is time to actually define methods that use our tools to generate meshes or prepare the data necessary to run the buckling analysis. These methods are the following:

- *SolveBucklingProblem(Owner, SP, out criticalLoads, out bucklingMode)*: This is the main method. It is in charge of preparing the data, call the solver and return the results to the user. The structure of the method is the following:
 1. *MeshParameter.cs* class is called to return the values that define the panel mesh based on the geometric properties of that panel.
 2. *SPmshBuildr.cs* class is used then to create the mesh using the previous parameters. First a new mesh object is created, and then the mesh is generated using the *CreateWholePanel* method explained in previous chapters.
 3. It is time to define loads and BC. First the values introduced by the user and stored in SP object are copied into new arrays. Then, in order

to translate these values into meaningful data for the solver, the class *INPUTdata.cs* is used.

4. Once the data is ready, the properties required by the solver to compute the solution are initialized and the values prepared in the previous steps are assigned. However, there is an extra transformation process that must be done at this step. Our SP object contains the material IDs used by the different entities in the stiffened panel, but it is necessary to use this ID to actually generate the ABDH matrix for the solver. In order to do so, four different laminates are created using that ID: SkinLaminate, WebLaminate, BottomFlangeLaminate and TopFlangeLaminate. However, it is very important to realize that the elements representing the bottom flange of the stringer actually represents the addition of the skin at the bottom and the bottom flange on the top. For that reason, it is necessary to create a fifth laminate that takes into consideration this issue. This fifth laminate is created by adding all the laminas from bottom to top: first the laminas contained in the skin are added to the new laminate, and then the laminas of the bottom flange are added consecutively.

Once the laminates have been defined, the matrices A (in-plane stiffness matrix), B (bending extension coupling matrix), D (bending stiffness matrix) and H (transverse shear stiffness matrix) are computed and their values are stored accordingly inside the ABDH array required by BMI3. It is also necessary to define the drilling factor, which does not influence much on the results but the solver needs it. It must have a value at about 100 times smaller than the diagonal terms in H matrix.

5. The solver is called through the intermediate class that was mentioned previously, which allows the comprehension between this C# code and solver's functions written in Fortran to find the primary bifurcation and return the corresponding eigenvalues and displacements.
6. Last step is to return the results so they can be used by the page and shown to the user.

The remaining methods are related to the visualization process:

- *DuplicateNodes(SP, ...)*: there are two different methods with the same name, one of them requires the displacements of the first buckling mode (and therefore is used to print the deformed shape) whereas the other does not require this displacements since it is used to draw the initial mesh. As it will be explained into detail in the visualization section, it is not possible to have one same node shared by different elements. It is necessary that each element contains 9 unique nodes. Since that is not what we have in our mesh, it is necessary to create the additional nodes required for the drawing. This means that in some points we will need to create several nodes with the same coordinates but different ID as they belong to different elements. The process followed by both methods is very similar:

1. Mesh is created following steps 1 and 2 of the previous method in order to get the panel's mesh. Namely, creation of a *meshparameters* object, creation of a *SPmshBuilder* object and mesh generation.
2. Initialization of the variables that will be returned from this method. These variables contain the new node's ID, new coordinate array, new displacement array and new element connectivity.
3. We actually start the process of generating the new variables. The procedure is the following: a loop is used to go through each element in the mesh. For each element, another loop will go through each of the nine nodes in that element. Then, for each node, a new ID is assigned starting from value 1, the coordinates of the nodes in the mesh are copied in the new array and, if the displacements are given, they are copied as well in a new array. In the new element connectivity array, the ID of this new node is added.

Basically this process allows nodes multiplication in the places where there was only one shared node. Let us imagine a node in the corner of an element. Due to the position of this element and to the fact that there are adjacent elements, this node in the FE mesh will be unique and shared by four adjacent elements in order to guarantee connectivity. In this step, nodes are created at that exact position, sharing the same coordinates and displacements, but they do have different ID. Hence, each of the nodes will belong to a different element.

4. Values of the new variables are returned.
- CreateGeometryToDraw(...): This method is used to prepare the variables required to print the mesh on the visualization page. The main arguments required are: the new node coordinates and displacements, and the new element connectivity that has been returned from the previous method, and the element entity array in order to identify the entity to which each element belongs to. This last variable is used in the visualization of the mesh in order to assign a different color to each entity, clarifying the identification of the different entities in the model. The rest of the arguments are, in fact, the variables returned by the method, which are: The coordinates of the nodes before deforming, the displacements of the nodes, the element connectivity of the triangles used to draw the model (different from the second order quadrangular element's connectivity obtained before), the vertices texture indices, the color array and the value of the factor that has been used to transform the coordinates into a 0 to 0.8 base in order to draw them in the canvas. Next the structure of the method will be presented and further information will be provided. It is important to understand that, even though part of the information has already been computed such as the nodes coordinates or displacements, they have to be presented in a specific format in order to be used by our visualization

tool (WebGL [58]). In case further details are demanded, consult the corresponding chapter where all this specifications are explained.

1. The variables are initialized.
 2. The viewport just shows nodes and entities which have coordinates ranging from 0 to 1. For this reason, we need to re-scale the whole mesh so that it fits in the canvas' view. In order to do so, coordinates of the first node in the mesh (which is the node at the corner formed by sides 1 and 4) are checked to identify the component with the largest absolute value. Then, this value is divided by a factor of 0.8. The result is the actual re-scaling factor. By dividing every coordinate and displacement by this value, the model is re-scaled. The maximum coordinate component of the model will be 0.8 (we left some margins to the viewport limits).
 3. The connectivity of the triangles that form the mesh is computed from the connectivity of the mesh elements. This process is further explained in the Visualization chapter.
 4. The texture array is computed assigning the corresponding pair of values for each node. There is further explanation in visualization chapter. This array is mainly used to draw the lines that identify the borders of the elements in the mesh.
 5. The color array which specifies the color for each node is computed. If the mesh is shown from My Documents, it just assigns a different color to each entity. On the other hand, if deformed shape of the panel is shown, it assigns a different color based on the magnitude of the total displacement of that node. More details are provided at Visualization chapter.
 6. The resulting variables are returned.
- GetFiles(...): this method stores in a variable all the content of some files that can be downloaded by the user, in particular an input file for Abaqus®, and two files (DAT and INP) for BMI3. The process is the following:
 1. Steps 1-4 mentioned in SolveBucklingProblem method are followed in order to generate all the variables required by any solver.
 2. Three variables are defined, each of which will contain the information of each file as a string. The content for both input files (for Abaqus® and BMI3) is computed directly by calling the methods corresponding methods of the *INPData.cs* class, and storing the information in the variables. In order to generate the DAT file required by BMI3, some code is added by which the information of the ABDH matrices is gathered and stored in the new variable accordingly to the format required by BMI3 to understand them (consult appendices B, C and D for further details). Finally the three variables are returned.

This is all the content of the new class *StiffenedPanel.cs*. In addition, all the code required to be added to *Kernel* project has already been explained.

7.2.2. KernelFortran Project: CADEC's solver

KERNELFORTTRAN is a Fortran project which contains all the functions necessary to carry out different analyses: buckling, damage, etc. It is the solver for all computations in CADEC regarding composites. For that reason, it was necessary to add all the source code of BMI3 in charge of computing the primary bifurcation and providing the buckling critical load and the first mode displacements. It is important to notice that, due to the difference in language, it is necessary to compile all this source code into a DLL (Dynamic library). Then, by means of the intermediate class, the DLL is imported and its functions can be accessed.

7.2.3. Web Project: CADEC's pages definition

WEB is a ASP.NET application project which contains all the web pages of the site. There are different folders inside this project, but only two of them contain the main pages that the user will see: *MyDocuments* and *Chapters*. The rest of the folders contain the pages for managing CADEC's account, administrator's pages, help pages, etc; pages that will not be affected by our application.

Before explaining the different pages that have been created, it is important to summarize how a web page is created with ASP.NET. Each page is composed by two different files that share a common name but have different extensions: the *.aspx* file and the *.aspx.cs* one. The former contains the information about the page and its layout, and can be written using ASP.NET elements, HTML elements, JavaScript code, etc. The latter is written in C# and is known as code-behind. This file contains the code that performs operations in the server, and actuates on the elements created in *.aspx* file.

Let us consider a simple example. Imagine that we want to create a simple website including a button and three textboxes. The intention of the page is to compute the sum of two numbers. The user would write two numbers in the first two textboxes, then he would press the button to calculate the result, and finally this result would be printed in the third textbox. In this case we would insert web elements (textboxes and button) in the *.aspx* file using HTML elements or ASP.NET ones. In this file, it is possible to decide the style of the elements (such as dimensions, borders, colors, ...), the layout of the page (their position in the page), etc. However, some code is required to read the values of the textboxes, performing the sum operation, and then show the new value in the third textbox. All this code would be written in the code-behind, inside a method that will be executed as a response to an event: it is triggered by clicking on the button. After clicking on the button, the page calls back to the server, executes the code inside the event, and then loads the page again.

A. My Documents

The following image shows the different pages that were created inside My Documents folder. These pages are: *Default* page, *Edit* page and *Visualization* page.

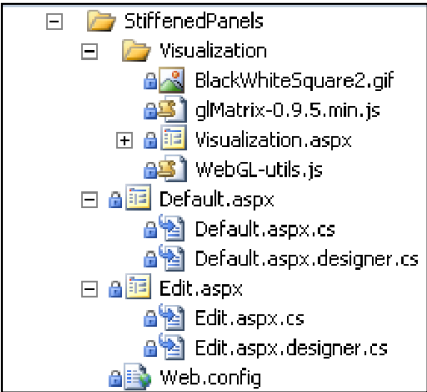


Figure 7.4: Files required for the pages in My Documents –CADEC-.

A.1. Default page

This page is accessed in the navigation tree by clicking on *My Stiffened Panels*. The layout of the page is shown in the image below:

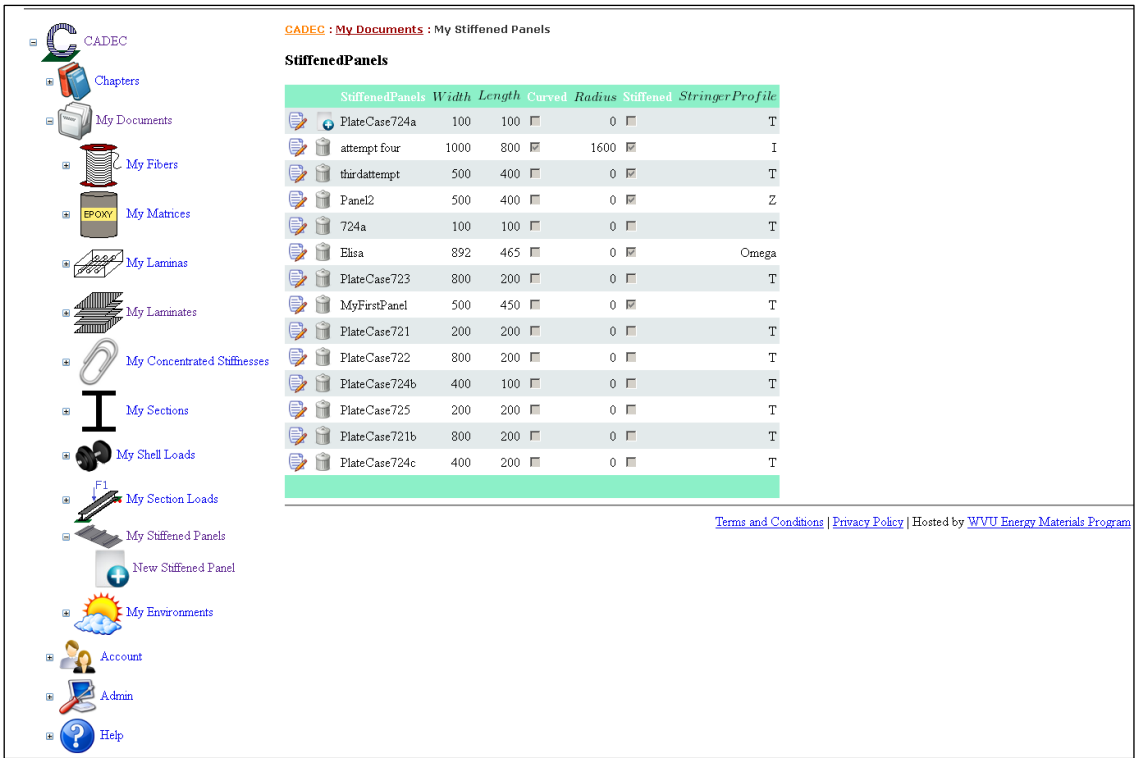


Figure 7.5: My Documents/Stiffened Panels/Default page of CADEC.

As it can be seen, this page shows all the panels that the user has in the database. The table includes the name of each stiffened panel, its two main dimensions (length and width), curvature information and stringer's profile used. In this case it is possible to identify each panel not only by name but thanks to the main properties described. On the left side of each name there are two symbols. The first one allows the user to edit that panel. In this case, the user will be redirected to the *Edit* page with the ID of the panel selected, so that all the properties of that panel are loaded correctly in the new page. The second symbol is a trash bin in all rows except the first one. By clicking on this element it is possible to delete a panel from the database. Finally, the second symbol in the first row performs the same operation as clicking on *New Stiffened Panel* in the navigation tree: the user is redirected to *Edit* page, but in this case without a panel ID since a new panel will be created.

A.2. Edit page

This page can be accessed in two ways, as it was explained before. One approach is by clicking on *New Stiffened Panel* in the navigation tree or in the second symbol in first row of *Default* page table. In this case the page will load some default values and the user will be able to create a new stiffened panel. The second approach is by clicking on the first symbol of any row in *Default* page, which allows the user to edit an existent panel. In this case, the properties of such panel are loaded in the page, and any modifications will be updated in that panel in the database. The Edit page layout is shown in the following image.

CADEC : My Documents : My Stiffened Panels : New Stiffened Panel

General Properties	
Name :	<input type="text"/>
Geometric Properties	
Width (A):	<input type="text" value="500"/> [mm]
Length (B):	<input type="text" value="400"/> [mm]
<input type="checkbox"/> Add curvature to the panel	

Stringer Properties	
Stringer Profile:	<input type="text" value="None"/>
Laminates	
Skin:	<input type="text" value="thirdattempt_SkinPlusBotFlangeLam"/>
Web:	<input type="text" value="thirdattempt_SkinPlusBotFlangeLam"/>
Top Flange:	<input type="text" value="thirdattempt_SkinPlusBotFlangeLam"/>
Bottom Flange:	<input type="text" value="thirdattempt_SkinPlusBotFlangeLam"/>
Loads	
BC and Load Type:	<input type="text" value="SkinOnly"/>

Side 3

N_r : [$\frac{N}{mm}$]

N_s : [$\frac{N}{mm}$]

BC:

Side 2

N_r : [$\frac{N}{mm}$]

N_s : [$\frac{N}{mm}$]

BC:

Side 4

N_r : [$\frac{N}{mm}$]

N_s : [$\frac{N}{mm}$]

BC:

Side 1

N_r : [$\frac{N}{mm}$]

N_s : [$\frac{N}{mm}$]

BC:

[Terms and Conditions](#) | [Privacy Policy](#) | Hosted by [WVU Energy Materials Program](#)

Figure 7.6: My Documents/Stiffened Panels/Edit page of CADEC.

The page has been rendered in order to create a new stiffened panel, so these are the default values. As it can be seen, the structure is very intuitive: introduce a name for the panel, then define the main dimensions, thirdly choose the materials for each entity in the panel and finally provide load and BC information. After all the input there are two additional rows. The first one contains a button that redirects the user to the visualization page in order to visualize the mesh. In order to avoid the problem of losing any changes made to the current panel, if the user clicks this button the information of the panel will be saved before redirecting the user to the visualization page. Otherwise the modifications would be lost and after coming to the page again it would only remember which panel was being edited, but it would load the old properties from the database. Finally, there the last row contains a save button which precisely allows the user to save the panel in the database. Being it a new panel, it is created

and saved. If the panel already exists, it updates the properties on the database. Just after all this table there is an additional row that allows the user to download a MSH file in order to visualize the mesh using gmsh [21]. This may be important in case the user's browser does not support the visualization technology WebGL [58].

The whole table is placed inside an update panel, which allows to have asynchronous post backs so that only the content of page defined in this panel is updated after any event, which improves the response of the page. Furthermore, the page is dynamic, meaning that some of the elements are updated as a response to a decision made. For example, by clicking on the checkbox "*Add curvature to panel*", a new row is added allowing the user to introduce the radius. Furthermore, the panel images are replaced by their counterparts:

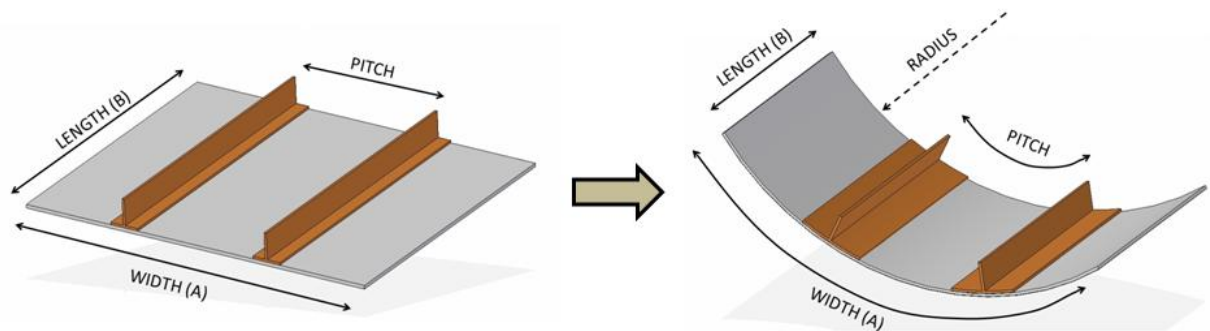


Figure 7.7: Layout of flat and curved stiffened panels containing the main dimensions. Created using Solid Edge®.

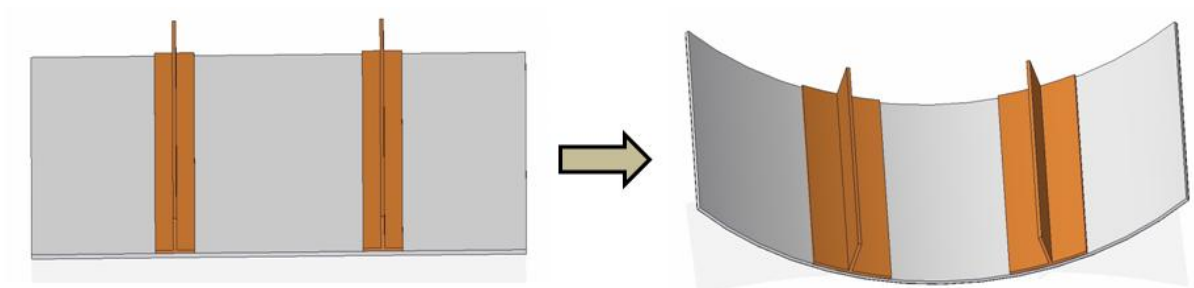
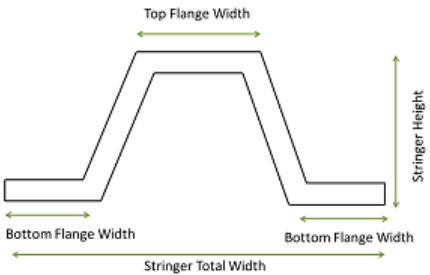


Figure 7.8: Flat and curved stiffened panel images. Created using Solid Edge®.

In addition, when the user changes the stringer's profile, the sketch of the profile is replaced by the corresponding image as well as the new rows necessary to introduce the required parameters for stiffened panels. Moreover, were *Omega* profile to be selected, two additional rows would appear since this profile requires four dimensions instead of two in order to be completely defined. The different profile layouts were shown in figure 2.5.

The next figure shows the additional information that the user needs to introduce when the panel is stiffened.

Stringer Properties	
Stringer Profile:	Omega
Stringer Pitch:	100 [mm]
Stringers Offset:	0 [mm]
Number Of Stringers:	1
Stringer Height:	25 [mm]
Stringer Width:	35 [mm]
Top Flange Width:	10 [mm]
Bottom Flange Width:	10 [mm]



The diagram illustrates the geometry of an Omega stringer. It shows a cross-section with a central web and two flanges. The 'Top Flange Width' is the width of the upper flange. The 'Bottom Flange Width' is the width of the lower flange. The 'Stringer Height' is the vertical distance between the top and bottom flanges. The 'Stringer Total Width' is the sum of the top and bottom flange widths.

Figure 7.9: Stringer properties definition in CADEC, showing the “Omega” profile example.

The offset property allows the user to move all the stiffeners with respect to the central position. If the offset is zero, the program computes the dimensions so that both lateral skins have the same width, which is equal to f_x value mentioned in Chapter 5. However, if we introduce a positive offset, the stringers will be moved towards the side 2 of the panel (towards positive Y direction), resulting in a wider skin on the left side and a narrower skin on the right side. The following image illustrates this behavior:

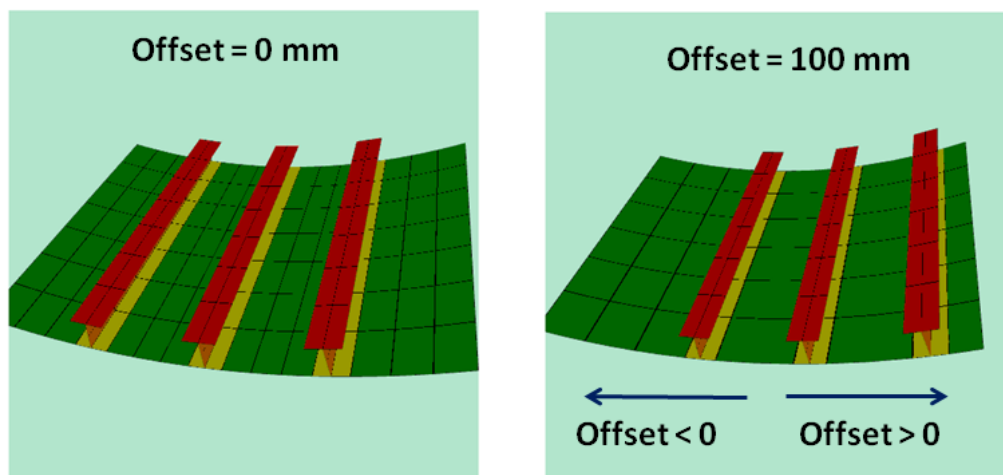


Figure 7.10: Different offset configurations. Left figure shows a stiffened panel with offset zero whereas right figure shows a positive offset. Created using CADEC output.

Finally it is important to notice that the current application is not capable of creating panels without lateral skins (lateral width equal to zero) or without intermediate skins (whenever the number of stringers is higher than two). In those cases the user should define dimensions in order to keep those parts as small as possible, but ensuring that the elements are not extremely deformed (high aspect ratios should be avoided, otherwise the accuracy of the solution could be strongly affected).

Finally, it is important to mention how the page has been created. In *aspx* file, each row has been created independently in *WebReusables* as it will be explained later. Therefore, it is just necessary to add a single line of code to insert each of these elements as rows in the table. This way of proceeding provides several advantages. For example, the validation of the values introduced by the user is carried out here in this page, using an element named *ValidateRows*. The code of this element is also contained in *WebReusables*. This element goes through all the elements (currently visible) inside the table and checks whether the values introduced are valid according to some properties that have been already defined for each row. This will be illustrated later on, but it is relevant to highlight that by considering each row as a different element to insert in the *aspx* file, it is possible to define there the range of values that the variable can take for example, among other properties. Then, special validation code is required in the code-behind, which makes the program code much better structured and clean. There may be some special cases in which logic validation is required (for example, it is not possible to apply Symmetry BC to a side and apply a load to that side). Such special situations do require explicit code in the code-behind.

Regarding the code behind of this page, it is quite straight forward. First, when the page loads some code is executed. This code is just executed the first time that the page is rendered (not a postback). Basically the code initializes the values of the textboxes or lists depending on whether a new panel is being created or a panel is being edited. Second, the definition of different event handlers can be found, which are methods that react to a specific event, and they perform the following actions:

- Save Click: This event handler is triggered when the user clicks on the Save image at the end of the table. The code inside is quite simple. First it checks whether all the values introduced by the user are valid (in type and range). If they are not valid, it does not perform any operation. If they are valid, then it executes some code. This code checks whether the user is creating a new panel or a panel is being edited. Depending on that, it calls the *Update* or the *Insert* methods that were previously explained in *StiffenedPanel.cs* class. Finally it checks whether these operations were carried out successfully in the DB.
- Visualize Click: This method is executed when the user clicks on the Visualization button. As in the previous example, it validates the fields in the table before actually running the remaining code. If the input data is valid, it creates a new stiffened panel object, updating all the geometric properties with the values in the fields (material, load and BC properties are not necessary for visualization). After that, the variables that will be sent to the visualization page such as node coordinates array, colors array, texture, etc. are declared. Then, the *DuplicateNodes* method is called, followed by a call to the *CreateGeometryToDraw*

method (both are stiffened panel class methods). These methods have been already explained previously in this chapter. Finally, since the response is going to be redirected to another page for visualization, it is important to store the values of these variables somewhere. Otherwise, visualization page would not be able to access the data. In order to do so, the variables are saved in current “Session”. The variables that need to be stored are: panel’s ID, its name, coordinates array, displacements array (which in this case are zero), texture vertices array, triangles connectivity array, colors array and the resize factor. Finally, once the variables have been stored in “Session”, the response is redirected so that the *visualization.aspx* page is loaded.

- *ShowCurvatureData*: it is in charge of showing the radius row in case the checkbox for curvature addition is checked. Moreover, it changes the images shown to the user from flat panels to curved ones.
- *RadiusChanged*: it computes the value of the angle form by the arc described by the stiffened panel, and finally it validates all the fields. The reason is that, without this limitation, it would be possible for the user to introduce a combination of panel dimensions and radius such that the resulting geometry could be almost a complete cylinder, or even more than 2π radians which would be impossible. Then, the length of the resulting arc must be restrained. The limitation imposed is that the angle covered by the arc could be, at the most, π radians (half of a cylinder). This code is executed whenever the content of the Update Panel is actually updated.
- *StringerTypeChanged*: this code is executed whenever the user changes the stringer’s profile for the model. The code basically identifies which is the new profile selected, and updates the image to the corresponding sketch. Furthermore, in case *Omega* profile is chosen, it also adds the two extra rows for introducing top and bottom flanges’ widths.

This is all for the Edit page. Now it is time to explain the Visualization page.

A.3. Visualization page

There will be two different visualization pages: one that will draw the mesh of the panel and a second one that will draw the deformed panel. The former page will be accessed from My Documents, from the previously explained Edit page. The latter will be accessed from My Chapters, after the solution of the problem has been computed.

Hence, in this case the page will be accessed by clicking the Visualization button in the Edit page. Since the Edit page does not solve the buckling problem, the purpose of this visualization page is to draw the mesh based on the geometric parameters introduced by the user. Displacements were thus set to zero. A different color will be applied on the elements belonging to each entity: skin, bottom flange, web and top flange.

The layout of the page can be seen in the following figure:

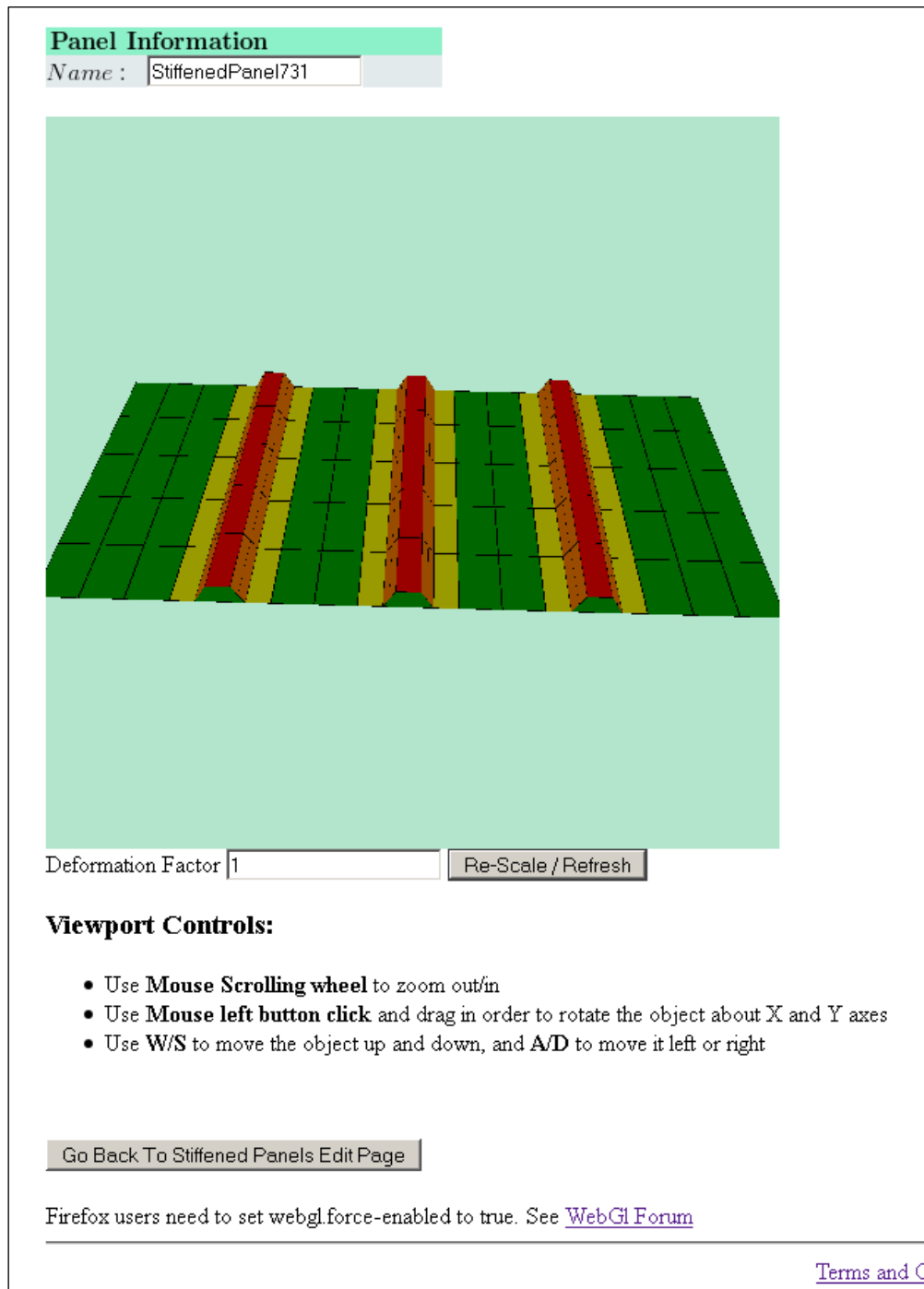


Figure 7.11: My Documents/Stiffened Panels/Visualization page of CADEC.

First, the name of the panel is displayed. Just after that, the canvas element is placed. As it is explained in the Visualization chapter, this canvas is the viewport where the scene will be drawn. In our case, the mesh of a panel will appear. There is a textbox under the canvas where the user can apply a different re-scaling factor by just typing the value and clicking on the button next to it. Since the displacements are zero in the first page, the purpose of this

element is not re-scaling but just refreshing the viewport in case a problem takes place. After that, a description about how to control and manage the object in the viewport can be found. Finally, there is a button in order to return to *Edit* page. The advantage of using this button instead of the navigation tree is that it will reload the panel in which the user was working on.

Regarding the code of the pages, it differs considerably when compared with previous pages, specially the *.aspx* file. The reason is that this page contains a lot of client-side code for the visualization, code written in JavaScript. Further details of this code can be found in the Visualization chapter. The purpose here is just to describe the overall code structure of the page. Basically the first lines are used to import two JavaScript libraries used by WebGL [58] for drawing the mesh. Then WebGL code comes inside a script tag. Finally, the real HTML and ASP.NET code can be found, which are responsible for the layout of the page.

The code-behind of the page is very simple. When the page loads, it retrieves the variables from “Session” and assigns these values to global variables. It is important to highlight that these global variables must be public so that JavaScript code can actually access them in order to draw the scene. Then, it fires the JavaScript function *webGLStart()*, which actually draws the scene inside the canvas element. Furthermore, it retrieves the text from resources and stores it in additional global variables so that the HTML elements can access it in order to print it.

B. Chapters

The following image shows the different pages that were created inside *Chapter's* folder. These pages are: *Default* page, *Buckling* page and *Visualization* page.

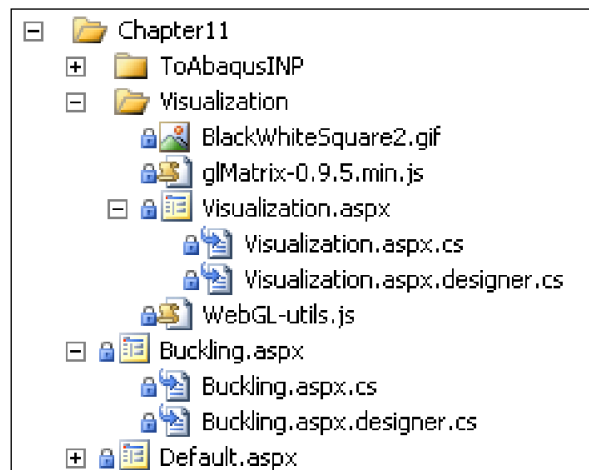


Figure 7.12: Files required for the pages in Chapters -CADEC-.

B.1. Default page

The *Default* page just contains an explanation of stiffened panels and buckling analysis, as it can be seen in the following image. This structure is followed in the rest of chapters inside CADEC, where a default page provides the theoretical base to understand the computations behind the code, and then other pages actually perform the calculations.

B.2. Buckling page

Buckling page shares the layout employed in *Edit* page. This is a good example of code reuse, since most of the controls in the table were already created for the *Edit* page of *MyDocuments*. The following image shows the page appearance.

CADEC : Chapters : Stiffened Panels : Buckling

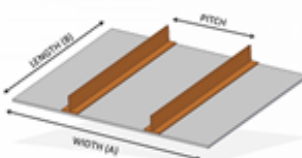
Buckling Analysis

Stiffened Panel

Stiffened Panel:

Geometric Properties

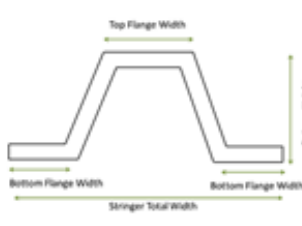
Width (A):	892 mm
Length (B):	465 mm
Stringer Pitch:	223 mm
CornerRadius (Radius):	0 mm



Stringer Properties

Stringer Profile:

Number Of Stringers:	3
Stringer Height:	26 mm
Stringer Width:	120 mm
Top Flange Width:	30 mm
Bottom Flange Width:	30 mm



Laminates

Skin:	SpecialtyOrthotropic
Web:	Case731_matC
Top Flange:	Case731_matC
Bottom Flange:	Case731_matC

Loads

BC and Load Type:

Buckling Load Case:

☒ Normal Loads

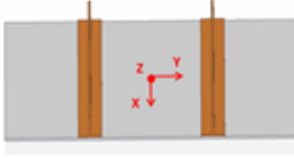
☐ Shear Loads

Side 3

N_x : $\frac{N}{mm}$

N_y : $\frac{N}{mm}$

BC:



Side 2

N_x : $\frac{N}{mm}$

N_y : $\frac{N}{mm}$

BC:

Side 4

N_x : $\frac{N}{mm}$

N_y : $\frac{N}{mm}$

BC:

Side 1

N_x : $\frac{N}{mm}$

N_y : $\frac{N}{mm}$

BC:

Download files

[Terms and Conditions](#) | [Privacy Policy](#) | Hosted by [WVU Energy Materials Program](#)

Figure 7.13: Chapters/Stiffened Panels/Buckling page of CADEC.

As it can be seen in the previous picture, the first element of the table is a dropdown list where the user can choose the panel that is going to submit for analysis. Then, the rest of the fields in the page are updated with the values of the panel properties. As in Edit page, the page is dynamic so images change depending on the curvature of the panel or the profile chosen. However, there are some modifications in the loading properties area. The application is not capable of solving problems with normal and shear loading at a time due to how the BC have been defined. Therefore, it is possible to run one loading mode at a time. If selecting normal loading (compression), the fields containing the shear values adopt a grey color, and the application will assume that their values are zero in order to guarantee that only normal forces are applied. If we choose shear loading, normal loading fields are disabled. Furthermore, if Symmetry is applied as BC in any side, automatically both loading fields are disabled since it is not possible to apply load under that kind of BC. All this is managed from the code behind, in an attempt to reduce the possible errors in the introduction of data by the user.

At the end of that first table there is a button called “Calculate”, which submits the problem to analysis. At this point, in order to inform the user that the calculation process is still running, a message is displayed just below the button as shown in the next figure. The orange loading bar is animated so that the user can see that the server is still working. This was necessary since some of the problems take more than a minute to be solved, and otherwise it is difficult for the user to identify whether the page went frozen or it is still waiting for the response of the server. After the problem is being solved, the user is redirected to the visualization page where results are presented.

The screenshot displays a web application interface for a structural analysis problem. It features four input sections for different sides of a structure:

- Side 3:** N_r : 0.069 $\frac{N}{mm}$, N_s : 0 $\frac{N}{mm}$, BC: SS
- Side 4:** N_r : 0 $\frac{N}{mm}$, N_s : 0 $\frac{N}{mm}$, BC: SS
- Side 2:** N_r : 0 $\frac{N}{mm}$, N_s : 0 $\frac{N}{mm}$, BC: SS
- Side 1:** N_r : 0.069 $\frac{N}{mm}$, N_s : 0 $\frac{N}{mm}$, BC: SS

Each section includes a text input field for the normal force (N_r), a text input field for the shear force (N_s), and a dropdown menu for the boundary condition (BC). A central diagram shows a rectangular structure with vertical orange bars representing loading on the sides. Below the input sections is a "Calculate" button. A green progress bar is shown below the button, and a message "Please wait while our crazy monkeys are trying to solve your problem..." is displayed next to it. At the bottom right, there is a "Download files" section with three buttons: "Download Abaqus INP file", "Download BMI3 INP file", and "Download BMI3 DAT file".

Figure 7.14: Loading bar appears while the problem is being solved on the server.

Additionally to the previous main table, there is a smaller table on the right side, which contains three buttons. They allow to download the corresponding files in case the user prefers to run the problem in another software such as BMI3 (for which it is necessary to download the *.inp* and the *.dat* files) or Abaqus® (*.inp* file).

Then, the *.aspx* file contains all the elements in the tables as in *Edit.aspx* file. Additional code is necessary to manage possible exceptions when trying to solve the problem, and also some code managing the loading bar that appears while waiting the response from the server.

Regarding the code behind of this page, it is quite straight forward. First, when the page loads some code is executed. This code is just executed the first time that the page is rendered, not after a postback. Basically the code initializes the values of the textboxes with the properties of the panel chosen by default in the dropdown list. Second, the definition of different event handlers can be found, which are methods that react to a specific event, and they perform the following actions:

- CurrentPanel PanelChanged: This method is called when loading the page the first time, or every time the user changes the selected panel of the dropdown list. It checks in session whether the user comes from visualization page, and in that case it chooses by default the same panel the user was working with before submitting the problem to be solved. Otherwise, it chooses the first panel of the list. Then all the properties are updated, and in some validation is performed such as disabling load fields of a side that has *Symmetric BC*.
- Bc Changed: this code is executed whenever the user modifies a BC, applying the corresponding restrictions to the loads depending on the type chosen.
- StringerTypeChanged: this code is executed whenever the user changes the stringer's profile for the model. The code basically identifies which is the new profile selected, and updates the image to the corresponding sketch. Furthermore, in case *Omega* profile is chosen, it also adds the two extra rows for introducing top and bottom flanges' widths.
- RadiusChanged: as in *Edit* page, it computes the value of the angle form by the arc described by the stiffened panel, and finally it validates all the fields. The limitation imposed is that the angle covered by the arc could be, at the most, π radians (half of a cylinder). This code is executed whenever the content of the Update Panel is actually updated.
- Calculate: This event is triggered when the user presses the Calculate button at the end of the table, and it contains most of the code of this page. After verifying that the input data introduced by the user is valid, it creates a new *StiffenedPanel* object that will define the problem. There is some code to manage this assignment. For instance, if the user's intention is to perform a buckling analysis under normal loading, the fields with the shear loading values are not read. Once the panel object has been updated, new variables are defined to store the results of the problem. At this point three methods of the class *StiffenedPanel.cs* are called. First, *SolveBucklingProblem* method is called to find the solution of the problem. Once it finishes, the output is retrieved. The eigenvalues are copied directly into the new variables, but the displacements need some preparation in order to be displayed later on in the visualization page. Therefore, *DuplicateNodes* method is called as well, followed by the method *CreateGeometryToDraw*. Finally all the data is ready to be used in the *Visualization* page, so the variables are saved in "Session" and then the user is redirected to the visualization page.

- *ClearResults*: it cleans the Session objects that were created to pass the results to the *Visualization* page. It is called once just after the Buckling page loads, once all the information contained was already used to identify the panel that the user was studying before going to the Visualization page.
- *DownloadAbaqusINPFile*, *DownloadBMI3INPFile* and *DownloadBMI3DATFile* share the same structure, and are in charge of writing on a file all the information required by the corresponding software to solve the problem. First they check whether the data introduced by the user is correct or not. Then, it creates a panel and updates it with the data of the user, but similarly to the *Calculate* event, there is some code to manage the exact data that is updated from the fields and the one that is not taken for that analysis. Then, the method *GetFiles* from *StiffenedPanel.cs* is called, which returns the content of each file in a variable. Then, depending on which file is required, the specific variable is written in a file so that the user can download it.

This is all for the *Buckling* page. Now it is time to explain the *Visualization* page.

B.3. Visualization page

In this case the page will be accessed by clicking the *Calculate* button in the *Buckling* page, just after the solution of the problem has been computed. The layout of the page can be seen in the following figure.

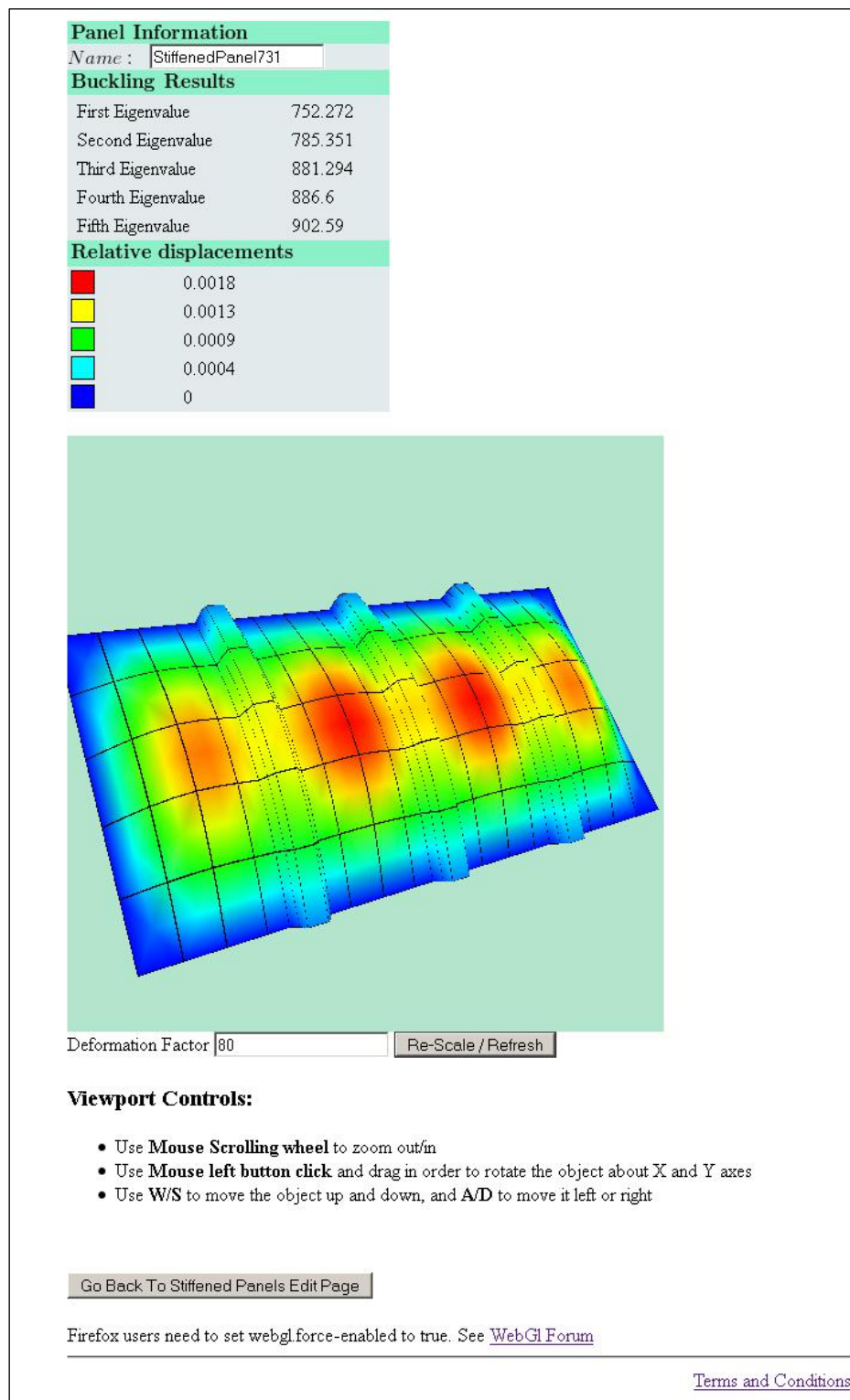


Figure 7.15: Chapters/Stiffened Panels/Visualization page of CADEC.

First, the name of the panel is displayed. Just after that, the values of the eigenvalues corresponding to the first five buckling modes are presented. By multiplying these eigenvalues times the load applied, the critical load can be obtained. Then, there is a legend containing the relative values of the displacements that correspond to each color in the viewport. Below that,

we can find the canvas element. As it is explained in chapter 8, this canvas is the viewport where the scene will be drawn. In our case, deformed panel corresponding to the first buckling mode is displayed. There is a textbox under the canvas where the user can apply a different re-scaling factor by just typing the value and clicking on the button next to it. In that way it is possible to exaggerate the displacements or attenuate them in order to clarify the view. After that, a description about how to control and manage the object in the viewport can be found. Finally, there is a button in order to return to *Buckling* page. The advantage of using this button instead of the navigation tree is that it will reload the panel in which the user was working on.

Regarding the code of the pages, it is almost the same as in the *Visualization* page accessed from *MyDocuments*. The *.aspx* file just contains some additional lines in order to define the tables above the canvas, but all the JavaScript code is the same.

The code-behind of the page is the same as well, except that in this case the colors are computed when the page loads by calling the method *SetColorArray*, instead of using the colors from Session (which would be the colors corresponding to different entities and not based on the deformed shape). This method does the following:

1. Since the displacements provided by the solver are given by components (u, v w), it is necessary to compute the module of the total displacement vector resulting of the addition of the previous components:

$$displacement_{node\ i} = \sqrt{u_i^2 + v_i^2 + w_i^2}$$

2. Go through each node in the mesh to find the maximum and the minimum total displacement. Then the range (which is the difference between the previous two values) is computed.
3. With the help of a loop it goes through all the nodes of the mesh. For each node, a percentage value is computed by the following formulae:

$$percentage = 100 \cdot \frac{displacement_{node\ i} - minimum\ displacement}{range}$$

4. Then, depending on the value, a color is assign as follows:
 - [0 , 25] %: Dark to Light Blue, from RGB(0,0,1) to RGB(0,1,1). Therefore, increasing linearly in this interval the component of green color.
 - (25 , 50] %: Light Blue to Green, from RGB(0,1,1) to RGB(0,1,0). Thus, decreasing linearly in this interval the component of blue color.
 - (50 , 75] %: Green to Yellow, from RGB(0,1,0) to RGB(1,1,0). Therefore, increasing linearly in this interval the component of red color.
 - (75 , 100] %: Yellow to Red, from RGB(1,1,0) to RGB(1,0,0). Thus, decreasing linearly in this interval the component of green color.

5. The values of the displacements of the legend are computed, at the percentages of 0%, 25%, 50%, 75% and 100% using the maximum and minimum displacements as well as the range.
6. The values are returned.

This is all the important code contained within the *Visualization* page.

7.2.4. WebReusables Project: code reuse across pages

WEBREUSABLES is a C# project which contains all the web elements created in order to be used in several pages, driving code-reuse. This is with no doubt where most of new code was created in order to assemble the application inside CADEC's environment.

The project consists of different folders. Most of them correspond to a specific object type such as lamina, fiber or panel. Inside the folder, all the web objects that represent a property of that object are stored. For example, the next image shows the different classes that were created in order to describe a stiffened panel.

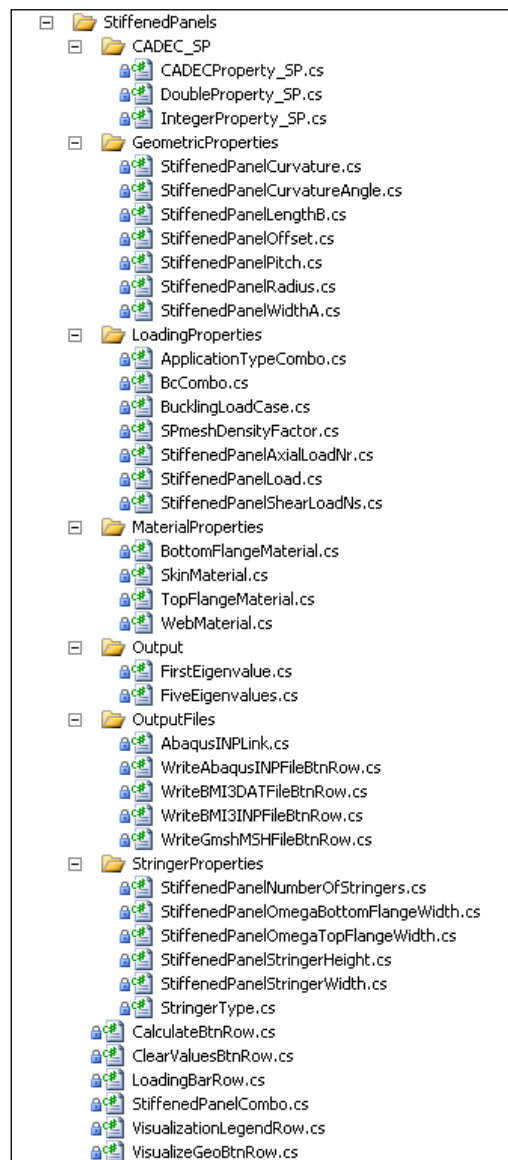


Figure 7.16: Files enabling code-reuse among the different pages related to Stiffened Panels in CADEC.

It can be seen that there are different subfolders inside Stiffened Panel's folder:

- CADEC_SP: It contains the code necessary to create a row in a table that will represent a property of an object. It is composed by different controls: one label (which include the name of the property), a textbox where the user introduces the value, and a third column that contains information about the units of that property. *CADECproperty_SP.cs* contains all these elements and specifies the layout, so that all the rows have a similar appearance. The other two classes are derived from this one, and add some code to explain that the property is going to be a *double* precision number, or an *integer*. They contain more code adding the properties that will define this type of Stiffened panel properties, which are basically:
 - Property name: text that will appear in the label.

- Property units: text that will appear in the units column
- Error message: text shown if the introduced value is not valid.
- Tip message: text shown when the mouse arrow is on the property label.
- Range error: text explaining a value-out-of-range error.
- Maximum value: maximum allowable value for the property.
- Minimum value: minimum allowable value for the property.

Then, after creating these general-purpose classes, it is easier to create new property classes. For example, the class defining the panel's width property can be created by the inheritance of the *DoubleProperty.cs* class (since the dimensions can be represented by a double precision number). Then, it is just necessary to assign the proper values to the previous properties, in this example:

- Property name: "Panel Width (A)".
- Property units: "mm".
- Error message: "The Panel's width value is not a valid number".
- Tip message: "panel's width dimension".
- Range error: "The panel's width should be a positive value".
- Maximum value: 10.000.
- Minimum value: 0.

Of course, all these values are not directly defined in this code. They are read from resources, so that it is easier to modify them or to translate them into different languages. This is the kind of way of proceeding that enhances the development and growth of the website, but which requires extra code.

- *GeometricProperties*, *LoadingProperties*, or *MaterialProperties*: contain different classes describing the main properties of the stiffened panel. Most of them inherit the previous Double or Integer properties' classes. The classes in charge of managing material selection replace the textbox by a drop-down list element. Therefore, this kind of classes contains different properties. One of the most complex classes is *StiffenedPanelLoad.cs* since it contains all the controls to create the following table layout:

The figure shows a graphical user interface for a stiffened panel analysis. In the center is a schematic diagram of a rectangular panel with two vertical stiffeners. Surrounding this diagram are four identical control panels, each corresponding to a side of the panel (Side 1, Side 2, Side 3, and Side 4). Each control panel includes:

- A header label (e.g., "Side 3" for the top panel).
- Input fields for N_r and N_s , each with a unit selector (either $\frac{N}{mm}$ or mm).
- A "BC" dropdown menu, currently set to "SS".

Figure 7.17: Part of a page in CADEC generated by a single file, illustrating the power of code reuse.

This class requires additional methods in order to access the individual controls inside. For instance, there are methods that retrieve the values of the normal and shear loads, as well as the BC applied; there are other methods to assign values to them, and there are additional methods to enable/disable some of the rows. This is necessary in order to restrain the user to introduce load values if “*symmetry*” BC is applied on that side, which would not be possible. Another example is that it is not possible to run a case with normal and shear loads at a time, so depending on the intended analysis just one kind of loading is allowed at a time. Moreover, it also contains a method to change the image displayed in case the user is working with a curved panel.

- Output and OutputFile: they contain the definition of the rows that display the eigenvalues, or the class that manages to download an INPUT file describing our problem that can be run in Abaqus®.
- The rest of the classes describe additional elements such as buttons, the loading bar that is shown while the problem is being solved, etc.

Apart from the previous classes, there are other classes in the folder *Images* in charge of managing the pictures display:

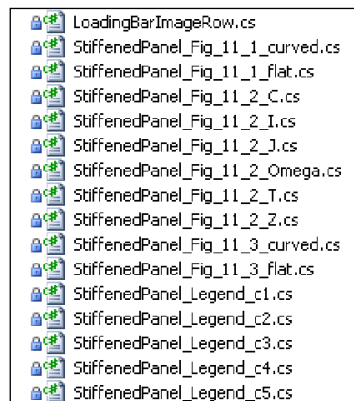


Figure 7.18: Files that manage the integration of images in a table row, used in the different pages related to stiffened panels in CADEC.

The code behind the classes is quite straight-forward. First a new class is created to set up the layout of the row. Then, in the actual class the actual name of the image as well as its location is added.

As it can be seen there is a lot of code in those classes, but it is not the aim of this project to explain every single line of code. Most of them use ASP.NET advanced programming. In case further details are desired, it is recommendable to consult the books referenced in the bibliography.

In addition, *WebReusables* project contains the resource files. It was necessary to store all the text appearing in the web (direct or indirectly) in resources so that it can be modified in the future or translated into different languages. Apart from the text, they also include the values defining the valid range for some of the properties. A resource file usually contains two

columns: the first column contains the name of the “variable”, and the second column contains its value or content. In that way, the variable is used throughout the code, and then its value is assign to other property it takes the value contained in the second column. In that way, it is much simpler to update a value or modify the content of a property.

Chapter 8

8. Visualization

8.1. The need of a visualization tool

At this point, the web application could be considered finished. It runs in a web environment, allowing the user to create a stiffened panel and subject it to some loads and boundary conditions, obtaining as a result the first five eigenvalues (and by multiplying it by the load, the buckling load can be easily computed). However, the buckling load is not completely representative by its own, especially in the design step. It indicates the load value at which the primary bifurcation takes place, when the instability occurs. Nevertheless, additional information regarding the panel deformation of at least the first mode could be very representative, providing very useful information. Indeed, based on that deformed shape it is possible to identify the weakest areas of the panel so that the designer could try to add some stiffness to them. This can be achieved sometimes by modifying the pitch so that skin areas are not excessively wide, or by modifying the dimensions or type of the stringers to increase the panel stiffness.

The displacements of the first buckling mode are already computed by the solver. One approach could be passing these values to the client in a text or excel file, so they can be visualized in the client's computer using a freeware such as gmsh [21] (alternative used by

SPD) or a commercial software such as Abaqus® or ANSYS®. However, this process would consume a lot of time for the user. A better, much more challenging approach would be to embed a visualization tool directly inside the application, in the web environment. In that case the user does not require any additional software installed in the computer.

Therefore, in order to improve the application usefulness a visualization tool will be added. The first decision to be made is selecting a visualization technology suitable for our needs. After looking for existing visualization tools to be used on the web, none of them were good enough for our purpose: the visualization tool must be created almost from scratch.

8.2. WebGL technology

One of the most useful and universal technologies used today is WebGL [22]. It is a new web technology that brings hardware-accelerated 3D graphics to the browser without installing additional software. It is the JavaScript version of the 3D library OpenGL® ES 2.0, targeted as low-level 3D API for phones and other mobile devices. The language used is GLSL (OpenGL Shading Language) for shaders as well as JavaScript.

WebGL is based on specific hardware architecture. The architecture assumes that a machine's graphics hardware is separated from its CPU. The graphics hardware likely has separate memory from the CPU. The graphics processing unit (GPU) specializes in running multiple copies of a single program at the same time. Unlike a normal CPU, these programs must be small and simple, but a GPU can run many, many copies of these programs simultaneously, making it faster than a normal CPU for graphics tasks. With this sort of hardware architecture, one of the biggest bottlenecks for fast 3D programs is the communication between the CPU and GPU.

The general design philosophy of WebGL is based on the idea that this communication between CPU and GPU should be minimized. Instead of repeatedly sending instructions and graphics resources from the CPU to the GPU, all of this data should be copied over only once and kept on the GPU. This minimizes the communication that needs to happen between the CPU and GPU. The communication that occurs is batched together into clumps so that the GPU can work independently from the CPU.

This is important in order to have a quick response in our application. Though such a tool does not require to run on the client side, it is strongly recommended to guarantee good performance. Thus, if our server can send all the necessary information to the GPU in one step, afterwards the data communication with our server can be minimum and all the visualization can take place at the client side. Let us illustrate this great advantage with a simple example. If the user wants to move the panel to one side of the viewport, this will require a process of re-drawing the panel continuously while moving the nodes, so the pixels are printed in the new

position. If the communication with the server is slow, this process will make the image blurry and intermittent. On the other hand, let us consider that all the information is sent to the GPU (nodes position, connectivity, shaders information, etc.) including programs that manages this movement. In this case, the program could run independently from the server at the client's browser, and the process of re-drawing the panel after a translation or rotation could be much faster. Thus, it is only necessary to pass all the information once, and then all the process takes place at the client side.

Now that the technology has been explained in terms of architecture, it is necessary to explain how WebGL works and what information requires to draw the panel. First of all, it is relevant to highlight that WebGL operates as a low-level language. This means that most of the functionalities must be implemented from zero. For instance, all the functions managing the rotation or translation of the view must be defined. The following paragraphs provide basic information about how WebGL draws and what main functions must be defined.

8.2.1. Renderer pipeline of WebGL

The process starts with the creation of the vertex arrays (nodes). These are arrays that contain vertex attributes like the location of the vertex in the 3D space and information about the vertex' texture, color or how it will be affected by lighting (vertex normal). These arrays and the information they contain are created in JavaScript in one or more of these ways: processing files that describe a 3D model (for example *.obj* files), procedurally creating the data from scratch, or using a library that provides vertex arrays for geometrical shapes. In our case, since nodes coordinates are known from the mesh, these values will be passed from the server to the JavaScript code. Basically, to define a vertex just three coordinates are required. Therefore, the coordinates of all the nodes will be passed (following a specific order) from the server.

Then the data in the vertex arrays is sent to the GPU by feeding it into a set of one or more vertex buffers. When a rendering job is submitted, we also have to supply an additional array of indices that point to the vertex array elements. WebGL has three main entity types: points and lines for 2D and triangles for 3D. Most of the 3D objects that are drawn are actually a composition of small triangles. In our case, the connectivity of these triangles will be computed programmatically at the server and the passed to JavaScript code. However, there are no lines between triangles so a uniform surface is obtained. In order to add some clarity to the panel and information about the mesh, dark lines defining the border of each quadrangle of the mesh will be added. Each quadrangle will be composed of eight triangles. The reason why 2D lines are not used is because just one type of entity can be defined at a time in a shader, and using several shader programs considerably increases the difficulty of the visualization tool. Then, in order to keep things as simple as possible, a texture will be used for that purpose.

The GPU begins by reading each selected vertex out of the vertex buffer and running it through the vertex shader. The vertex shader is a program that takes a set of vertex attributes as inputs and outputs a new set of attributes. At a minimum, the vertex shader calculates the

projected position of the vertex in screen space. But it can also generate other attributes such as color or texture coordinates for each vertex. For our panels, both color and texture will be defined. Color will be computed in the server or when the page loads, depending on the page. If the code is run from *MyDocuments* in order to see the mesh, a different color for each entity in the panel (skin, bottom flanges, webs and top flanges) will be used for clarity. On the other hand, if page is ran from *MyChapters*, the intention is to visualize the deformed panel. Thus, the colors will be computed using a range of colors from dark blue to red, corresponding from displacement zero to the maximum value. As it was mentioned in the previous paragraph, a texture will be used to define mesh elements borders. This texture is a white square with a black frame at the borders. How all this works will be explained later on. All the data regarding the colors and textures will be passed to the JavaScript code.

The GPU then connects the projected vertices to form triangles. It does this by taking the vertices in the order specified by the indices array and grouping them into sets of three. The indices array, which contains the connectivity, is passed from the server as well.

The rasterizer takes each triangle, clips it, discards parts that are outside of the screen, and breaks the remaining visible parts into pixel-sized fragments. The vertex shader's outputs for other vertex attributes are also interpolated across the rasterized surface of each triangle, assigning a smooth gradient of values to each fragment. For example, if the vertex shader assigns a color value to each vertex, the rasterizer will blend those colors into an appropriate color gradient across the pixelated surface. This means that, at least, the color interpolation between nodes is handled by WebGL and luckily we do not need to manage it.

The generated pixel-sized fragments then pass through another program called the fragment shader. The fragment shader outputs color and depth values for each pixel, which then get drawn into the framebuffer. Common fragment shader operations include texture mapping and lighting (in our case lightning will not be added). Since the fragment shader runs independently for every pixel drawn, it can perform the most sophisticated special effects; however, it is also the most performance-sensitive part of the graphics pipeline. As with the vertex shader, you can code your own fragment shader or use one provided by a WebGL library.

The framebuffer is the final destination for the rendering job's output. A framebuffer is more than a single 2D image: in addition to one or more color buffers, a framebuffer can have a depth buffer and/or stencil buffer, both of which optionally filter fragments before they are drawn to the framebuffer. Depth testing discards fragments from objects that are behind the ones already drawn, and stencil testing uses shapes drawn into the stencil buffer to constrain the drawable part of the framebuffer, "stencilling" the rendering job. Fragments that survive these two filters have their color value alpha blended with the color value they're overwriting. Final color, depth, and stencil values are drawn into the corresponding buffers. The buffers' outputs can also be used as texture inputs to other rendering jobs.

The next image shows all the previous process and how does WebGL and the graphic processor create the final scene.

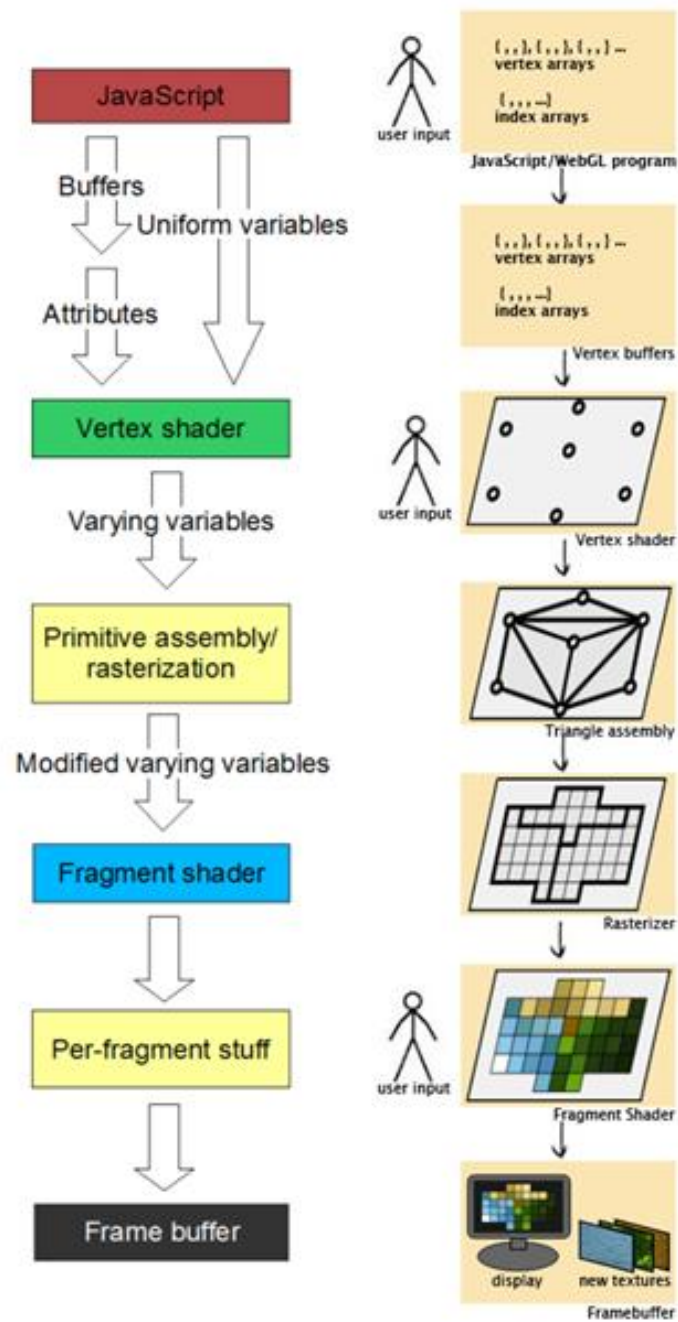


Figure 8.1: Two schemes representing the renderer pipeline of WebGL. Sources: [1] and [22].

8.2.2. Definition of the different variables used by WebGL

Then, there are some variables that must be computed at the server side and then passed to the JavaScript code so they are added to the rest of WebGL code. These variables basically are:

- **VertexPositionBuffer:** one dimensional array which contains the coordinates of each of the N nodes that define the panel. Data is arranged following the next sequence: $[X1, Y1, Z1, X2, Y2, Z2, \dots, Xn, Yn, Zn]$. It is important to mention that more nodes are required to draw the panel than those contained in the mesh. The reason is that, in order to apply the texture, it is not possible to have one single node shared by more than one quadrangle element. This occurs because in order to apply the texture, each vertex must have some information regarding how the texture is applied. Since the texture is applied separately to each element, one shared node would have to contain multiple information as it belongs to several elements, which is not possible. Therefore, the amount of nodes required will be the number of elements times nine nodes per element.

It is important to notice that the final coordinates are computed programmatically in JavaScript code, in order enable re-sizing of the deformed shape without requiring a postback. Then, two arrays are actually passed to JavaScript code: one for the initial coordinates, and a second one for the displacements. Both arrays are managed in the server so they are sent with the structure previously discussed. Then, in the client side code, the final coordinates are computed as:

$$[X, Y, Z] = [X_0, Y_0, Z_0] + RSfactor \cdot [u, v, w]$$

Where the subindex zero stands for initial values, RSfactor determines how displacements are rescaled, and u,v,w are the displacements in x,y,z respectively.

- **IndicesBuffer:** one dimensional array which contains the connectivity for the triangles to be drawn. This connectivity is defined by setting the connectivity inside an element, and then going through all the elements contained in the mesh. For instance, the connectivity for this element would be: $[1, 5, 9, 1, 9, 8, 2, 6, 9, \dots, 4, 9, 7]$.

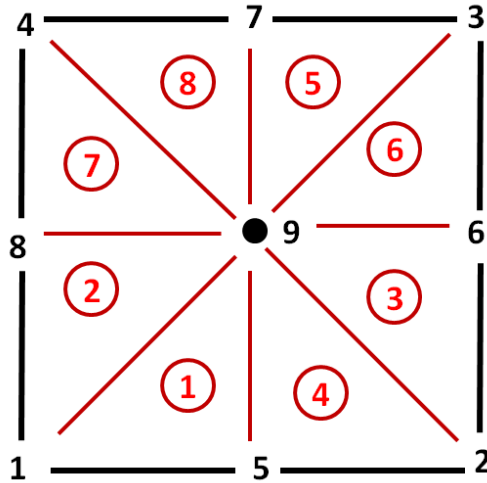


Figure 8.2: Scheme of a nine-node mesh element division into triangles for visualization.

- ColorsBuffer:** one dimensional array which contains the color information for each vertex. Colors are defined in WebGL using the RGBA color space, which means that four parameters must be defined for each vertex. The first three parameters are the corresponding amount of Red, Green and Blue that generates our color. These parameters range from 0 to 1. The final parameter, which also ranges from 0 to 1 is alpha, which is a measure of the opacity of the color. Alpha equal to 1 means fully opaque, whereas alpha equal to 0 would be transparent. As color is defined by vertex, the color array would have the following structure: $[R1, G1, B1, A1, R2, G2, B2, A2, \dots, Rn, Gn, Bn, An]$.
- TextureBuffer:** one dimensional array which contains the information about how the texture is applied to the elements. In our case, this texture is a white square with black frame. Then, in order to apply it to each element, it is necessary to tell WebGL how to match the pixels of the texture with the pixels in the elements. In this case we will define nine pairs of values per quadrangular element (one pair per vertex). What these texture coordinates specify is where, in cartesian x, y coordinates, the vertex lies in the texture. For the purposes of these coordinates, we treat the texture as being 2.0 wide by 2.0 high, so $(-1, -1)$ is at the bottom left, $(1, 1)$ the top right. The conversion from this to the real resolution of the texture image is handled for us by WebGL. The image below shows the texture coordinates that must be specified for each quadrangular element. All these coordinates are defined in order, from vertex 1 to vertex N , following the structure: $[X1, Y1, X2, Y2, \dots, Xn, Yn]$.

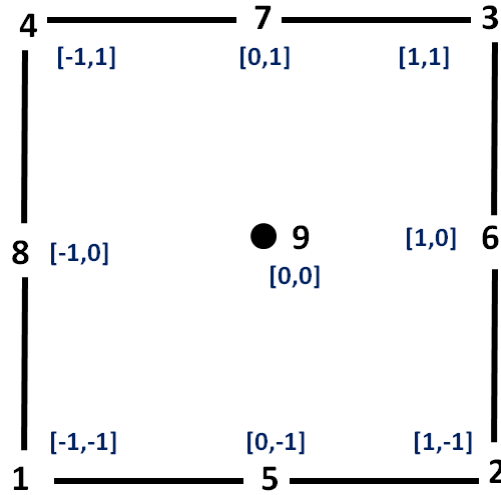


Figure 8.3: Texture coordinates for each node in a nine-node element.

The interesting thing about using this texture is that color black in RGB form is the absence of color (0, 0, 0), whereas white is the opposite (1, 1, 1). Then, if we define the final color for each pixel in the fragment shader as follows:

$$[R, G, B]_{final} = [R, G, B]_{color} \cdot [R, G, B]_{texture}$$

We are then able to draw dark lines in the border of the quadrangular element. When a pixel lays in the frame of the texture, the result of the previous operation is (0, 0, 0) –black color–, no matter which color we had previously. In addition, when a pixel lays on the white zone of the pixel, the resulting operation is the exact same color (multiplying by one does not alter the values). Hence, this work around allows us to draw black lines at the element borders without the need of defining multiple shader programs. The only minor drawback of this approach is that if the elements are too slim, the lines of the small edges are wider than those in the large edges. However, it is just a matter of aesthetics, which does not affect the main objective of our tool.

8.2.3. Description of the JavaScript code

Finally, it is time to actually describe our tool. Instead of talking about the code itself, a review of the structure of the code will be presented:

- i. The very first step is just including two separate JavaScript files (*.js*) in the page, so some of the functions defined there can be used. One of the files includes some utils for WebGL that will reduce the code to initialize it. The second file includes the definition of several operations with matrices such as rotation, translation, determinant, normalization,... These functions were separated into a different file as libraries in order to encourage code-reuse as well as reducing the code inside the visualization page.

- ii. Second, define the vertex and fragment shaders, along with their attributes. The code for each shader will be placed into a different script block.
- iii. Then, some code is required in order to initialize WebGL context. At this point we try to initialize WebGL inside the canvas.
- iv. After that, shades are loaded into WebGL program object, getting the shaders and initializing them.
- v. Define the model-view matrix and the projection matrix, along with a specific function that push them over the JavaScript/WebGL divide so that the shaders can be seen. The model-view matrix basically records all the transformations that we perform on the object, such as zoom, rotation or translation. The projection matrix is in charge of adding perspective to the object (elements further of the user view are smaller than those closer to the view), which provides realism to the object.
- vi. Next, some extra functions are defined in order to provide our application with interesting features that enhance the visualization of the object. These functions include all the code to answer to keyboard and mouse events. Thanks to this functionality, mouse wheel can be used to zoom in or out, left click and mouse movement allow the user to rotate the panel and, finally, by pressing some keys it is possible to move the panel as well. All together they form an excellent tool for postprocessing results.
- vii. Load up buffers containing information about the objects in the scene. At this point is when all the data from the server has already been embedded here, so that it is available for the GPU later on. The buffers include the previous variables that were mentioned before, such as vertex positions, connectivity and colors.
- viii. Next step is defining a function which actually draws the scene. This function defines the viewport dimensions and characteristics, applies the matrices over the object, and actually draws the triangles, vertices etc.
- ix. Define a function that will set everything up at the first moment (initialization). After the set up, it calls a loop that is drawing the scene repeatedly, and this loop will never stop. This is the reason why while rotating the object, the panel is being printed all along the process, in opposition to printing just the final position.
- x. Provide HTML code required to display it. The main HTML control that is going to be used is the canvas. The canvas element is somehow the viewport where the whole scene will be displayed.

Chapter 9

9. Tool validation

This chapter will test the accuracy of the results provided by our tool. First, using Classic Plate Theory some of the simplest cases whose approximate solution is known will be subject to analysis. Second, results obtained after analyzing a curved plate will be compared to those provided by commercial Finite Element software found in literature. Finally, in order to validate the application successfully, laboratory tests of stiffened panels subject to compression will be simulated with CADEC so results can be compared to experimental data found in the literature.

9.1. Validation with theoretical expressions of Classical Laminated Plate Theory (CLPT)

9.1.1. Flat Plates subject to compressive loads

A plate is a flat structural element having two dimensions much larger than the thickness. For most shell problems, the in-plane stress resultants N_x , N_y and N_{xy} can be obtained using only equilibrium equations. However, this is not the case for plates, in which moments M_x , M_y and M_{xy} cannot be found from the applied loads without specifying the material and the thickness. For this reason, in preliminary design some approximate methods which are valid only for particular cases are used. Normally this preliminary design is followed by a refined Finite Element Analysis. Since the aim of our tool is to help in the preliminary design, it should obtain similar results than those derived from the theoretical approximated expressions.

The first objective is to verify our tool's performance for analyzing flat plates under compressive loading. Different plate dimensions as well as BC and loading layouts will be considered, and the results will be compared to those computed with Classic Plates Theory. The analytical expressions have been obtained from the literature [7]. Since there are no stringers, BC and loads will be applied just on the skin.

The solutions predicted by the expressions found in the literature [7] are based on the assumption that the laminates are symmetric and specially orthotropic with $[B] = [0]$ and $D_{16} = D_{26} = 0$. For balanced symmetric laminates with non zero values for D_{16} and D_{26} , the equations may predict erroneous results.

The same laminate will be used for all these cases. It will be formed by glass/epoxy laminas with the following properties:

Lamina properties	
E_1 (GPa)	97.30
E_2 (GPa)	6.99
ν_{12}	0.28
G_{12} (GPa)	2.964

Table 9.1: Lamina properties for all flat plates cases.

The properties of the laminate are summarized in the following table:

Laminate properties	
Stacking sequence	[90/0] _s
Lamina thickness (mm)	0.1

Table 9.2: Laminate properties used in all flat plates cases.

The following theoretical expressions use the components of the [D] matrix -bending stiffness matrix-, which for the previous laminate are summarized in the next table.

Bending stiffness matrix components (Mpa mm ³)					
D_{11}	D_{12}	D_{22}	D_{16}	D_{26}	D_{66}
98.040	10.504	461.328	0	0	15.808

Table 9.3: Summary of the resulting coefficients of the bending stiffness matrix [D], which are necessary to compute the theoretical critical buckling load.

In order to ensure units coherency, dimensions will be introduced in millimeters, and loads in N/mm.

A. Flat plate with simply supported boundary

The buckling load per unit length of a flat plate simply supported around the boundary and subject to one direction loading can be computed with the following expression (expression 11.4 in [7]):

$$N^{CR} = \frac{\pi^2 \cdot D_{22}}{b^2} \cdot \left[m^2 \frac{D_{11}}{D_{22}} \left(\frac{b}{a} \right)^2 + 2 \cdot \frac{D_{12} + 2D_{66}}{D_{22}} + \frac{1}{m^2} \cdot \left(\frac{a}{b} \right)^2 \right]$$

Where m is the number of waves of the buckled shape along the loading direction. This number can be computed as the nearest integer to the real number R_m :

$$R_m = \left(\frac{a}{b} \right) \cdot \left(\frac{D_{22}}{D_{11}} \right)^{0.25}$$

The input data considered for this case is the following:

Geometric Parameters			BC	LOADS	
				Nr	Ns
Curvature	None	Side 1	SS	0	0
StringersNum	0	Side 2	SS	1	0
Width (A)	100	Side 3	SS	0	0
Length (B)	100	Side 4	SS	1	0

Table 9.4: Loads, BC and geometry input for plate A in CADEC.

Which result in a $R_m = 1.47$, so that $m=1$ (first mode). The solution predicted by the theoretical expressions is:

$$N^{CR} = 0.635$$

And the results obtained with CADEC are:

$$N^{CR} = 0.636$$

Producing the following deformation plot of the plate.

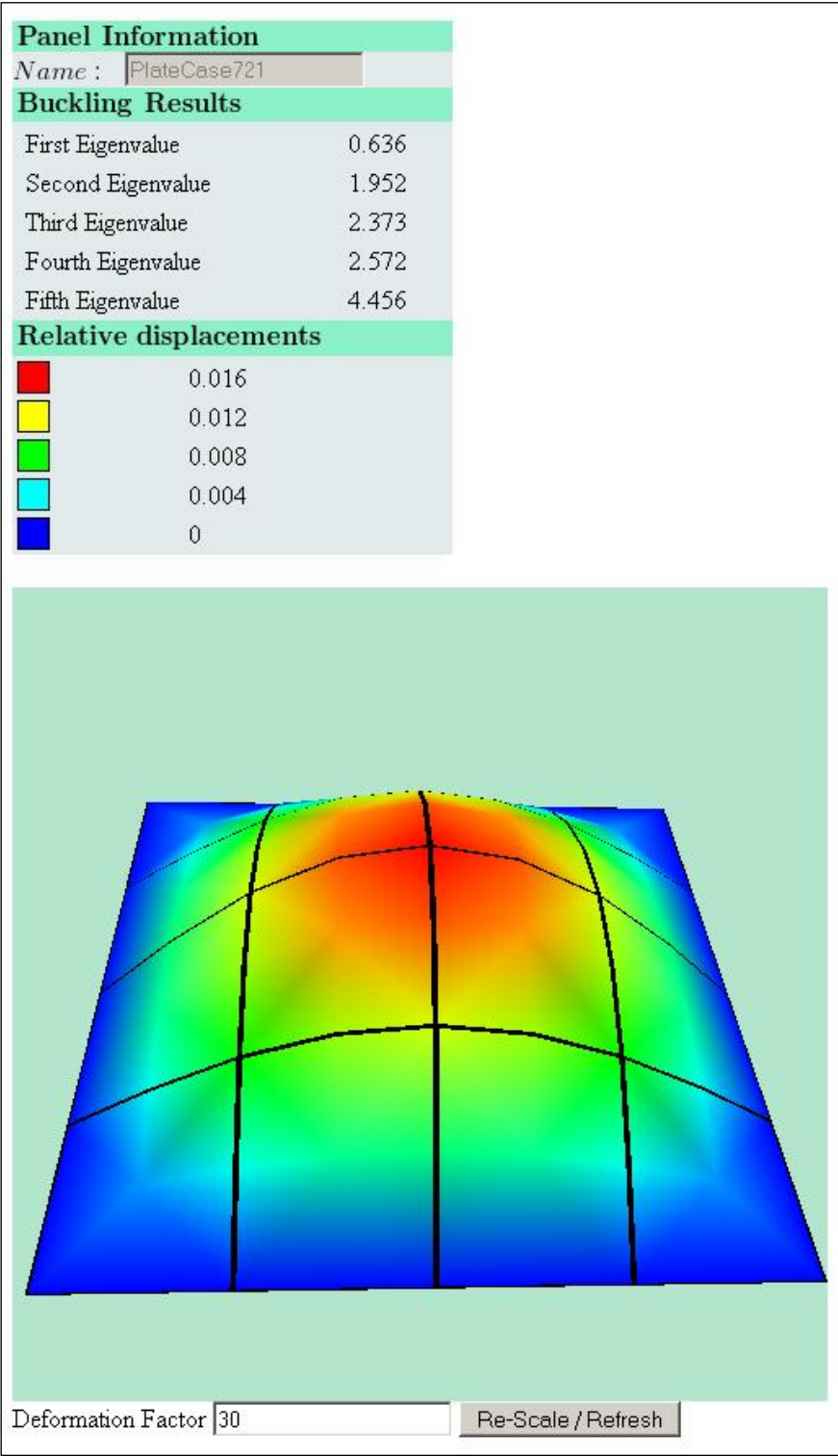


Figure 9.1: Buckling results for a flat plate with simply supported boundary subject to uniaxial compression using CADEC.

The solution predicted by our application for this case is almost the same as the one predicted theoretically, with a difference of 0.12%.

However, were the size of the plate such that $a \gg b$, then the buckling load could be obtained by using the following equation (equation 11.6 in [7]):

$$N^{CR} = \frac{2 \cdot \pi^2}{b^2} (\sqrt{D_{11}D_{22}} + D_{12} + 2 \cdot D_{66})$$

If $a=500$ and $b=100$, the solution predicted by the theoretical expressions is:

$$N^{CR} = 0.503$$

And the results obtained with CADEC are:

$$N^{CR} = 0.523$$

Leading to the deformation plot of the plate shown in next page.

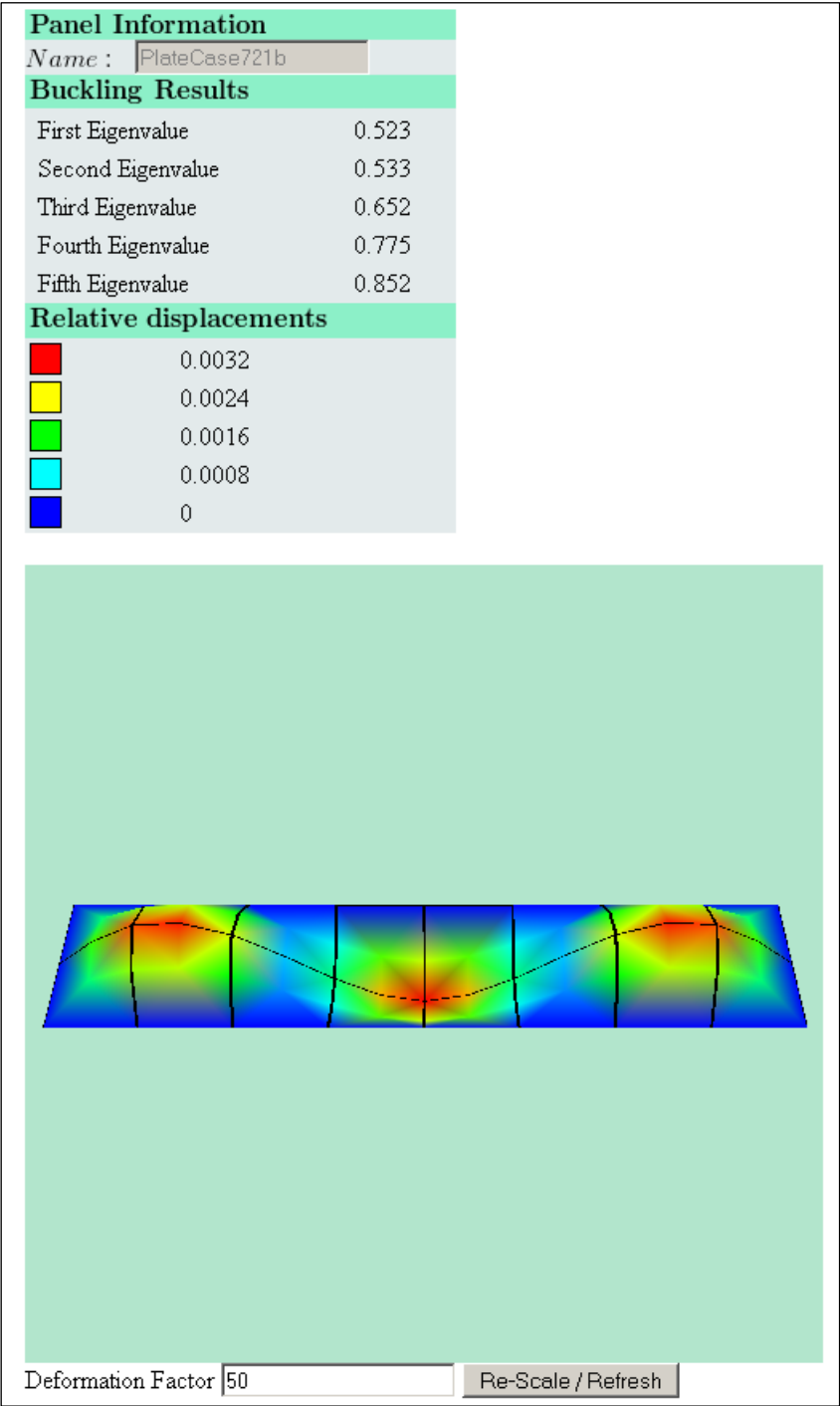


Figure 9.2: Buckling results for a slender flat plate with simply supported boundary subject to uniaxial compression using CADEC.

The difference in results for this case is 3.99%.

B. Flat plate with clamped boundary

The buckling load per unit length of a flat plate clamped around the boundary (but in-plane displacements enabled) and subject to one direction loading can be computed with the following expression (expression 11.11 in [7]) as long as the dimensions satisfy that $a/b > 4$:

$$N^{CR} = \frac{\pi^2}{b^2} \cdot [4.6 \cdot \sqrt{D_{11}D_{22}} + 2.67 \cdot D_{12} + 5.33 \cdot D_{66}]$$

The input data considered for this case is the following:

Geometric Parameters			BC	LOADS	
				Nr	Ns
Curvature	None	Side 1	CC	0	0
StringersNum	0	Side 2	CC	1	0
Width (A)	500	Side 3	CC	0	0
Length (B)	100	Side 4	CC	1	0

Table 9.5: Loads, BC and geometry input for plate B in CADEC.

Which result in a theoretical critical buckling load of:

$$N^{CR} = 1.076$$

And the results obtained with CADEC are:

$$N^{CR} = 1.217$$

Being the deformation plot of the plate shown in next page.

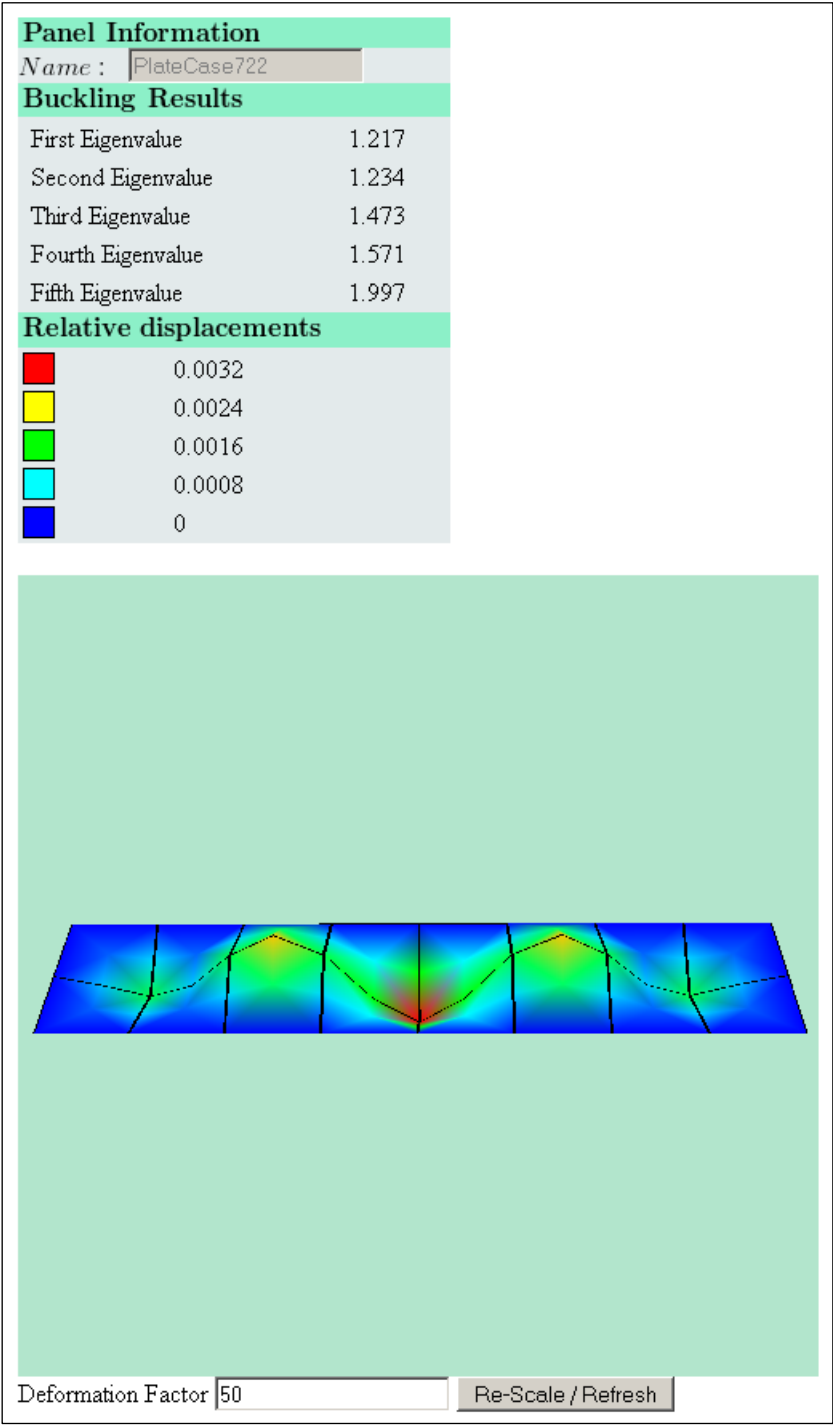


Figure 9.3: Buckling results for a flat plate with clamped boundary subject to uniaxial compression using CADEC.

It leads to a difference of 13.07%. Later on this case will be run analyzing just one fourth of the panel and applying symmetric BC. In that case, since the refinement of the mesh increases, the solution obtained converges to the theoretical value.

C. Resolution of previous cases applying symmetric BC

It is possible to apply symmetry BC to any side of the panel. This allows the user to analyze a smaller portion of mesh in order to obtain the same results. Given that the tool is conceived to be used in preliminary design, the mesh is quite coarse so the time saved by analyzing half a model or one quarter could not be so important. However, in finer meshes this methodology could provide almost the same results saving hours or even days in terms of time-consumption. Moreover, it is possible to spend the same amount of time for solving the problem but still get a more accurate result, since the same mesh represents one fourth of the model instead of its whole set.

C.1. Case A: flat plate with simply supported boundary

The input data considered for this case is the following:

Geometric Parameters			BC	LOADS	
				Nr	Ns
Curvature	None	Side 1	SS	0	0
StringersNum	0	Side 2	SYM	0	0
Width (A)	50	Side 3	SYM	0	0
Length (B)	50	Side 4	SS	1	0

Table 9.6: Loads, BC and geometry input for plate D.1 in CADEC.

For this model, the buckling load predicted by CADEC is:

$$N^{CR} = 0.635$$

Which can be compared to the results previously predicted by the theoretical expressions:

$$N^{CR} = 0.635$$

Generating the following deformation plot of the plate.

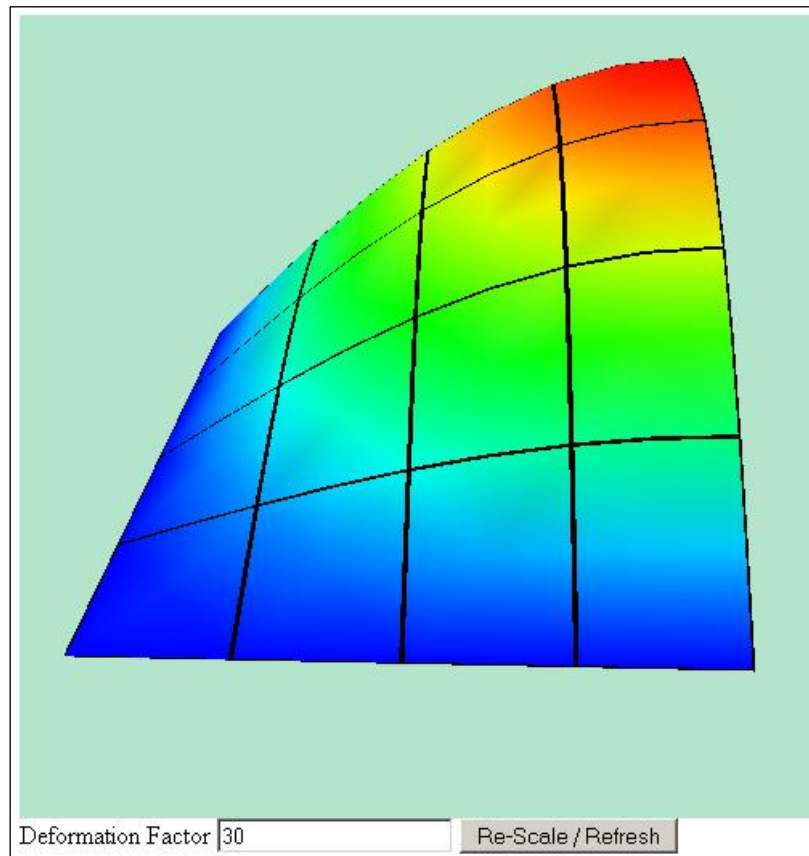


Figure 9.4: Deformed shape of a flat plate with simply supported boundary applying symmetric boundary conditions using CADEC.

The results in this case are just 0.03% different.

C.2. Case B: flat plate with clamped boundary

The input data considered for this case is the following:

Geometric Parameters			BC	LOADS	
				Nr	Ns
Curvature	None	Side 1	CC	0	0
StringersNum	0	Side 2	SYM	0	0
Width (A)	250	Side 3	SYM	0	0
Length (B)	50	Side 4	CC	1	0

Table 9.7: Loads, BC and geometry input for plate D.2 in CADEC.

For this model, the buckling load predicted by CADEC is:

$$N^{CR} = 1.127$$

Which can be compared to the results previously predicted by the theoretical expressions:

$$N^{CR} = 1.076$$

Generating the following deformation plot of the plate:

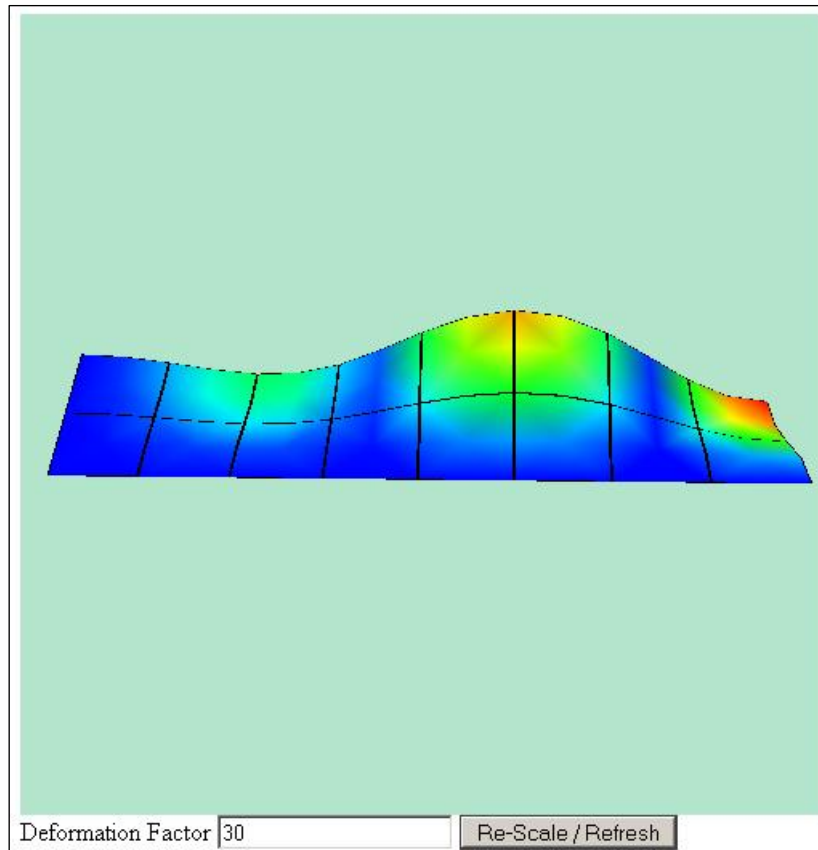


Figure 9.5: Deformed shape of a flat plate with clamped boundary applying symmetric boundary conditions using CADEC.

As we can see, since mesh has been refined, the difference has drop down from 13.07% when analyzing the whole panel to just 4.71% when focusing on one fourth of it, and applying symmetric BC. It is possible to see how the deformed shape of the panel is consistent with the one without symmetry: the buckling mode of the total panel was in the form of five half-waves in the largest direction, whereas in this case the model represents only two and a half half-waves in that direction (since only a quarter of the panel is being analyzed).

D. Flat plate subject to biaxial loading

In this case a flat plate with its boundary simply supported subject to compressive loading in directions x and y will be considered. The buckling load can be predicted by the next formulae (expression 11.15 in [7]):

$$N^{CR} = \frac{\pi^2}{b^2} \cdot \frac{D_{11}m^4c^4 + 2Hm^2n^2c^2 + D_{22}n^4}{m^2c^2 + kn^2}$$

Where,

$$c = b/a$$

$$H = D_{12} + 2D_{66}$$

$$k = \frac{N_y}{N_x} = \frac{N_y^{CR}}{N_x^{CR}}$$

The parameters m and n represent the number of half-waves in “ x ” (panel’s width direction) and “ y ” (panel’s length direction). The critical buckling load is obtained for the pair of values for m and n such that the previous expression’s result is minimum.

The input data considered for this case is the following:

Geometric Parameters			BC	LOADS	
				Nr	Ns
Curvature	None	Side 1	SS	1	0
StringersNum	0	Side 2	SS	1	0
Width (A)	100	Side 3	SS	1	0
Length (B)	100	Side 4	SS	1	0

Table 9.8: Loads, BC and geometry input for plate E in CADEC.

Which result in a theoretical critical buckling load of:

$$N^{CR} = 0.317$$

And the results obtained with CADEC are:

$$N^{CR} = 0.318$$

Leading to the deformation plot of the plate shown in next page.

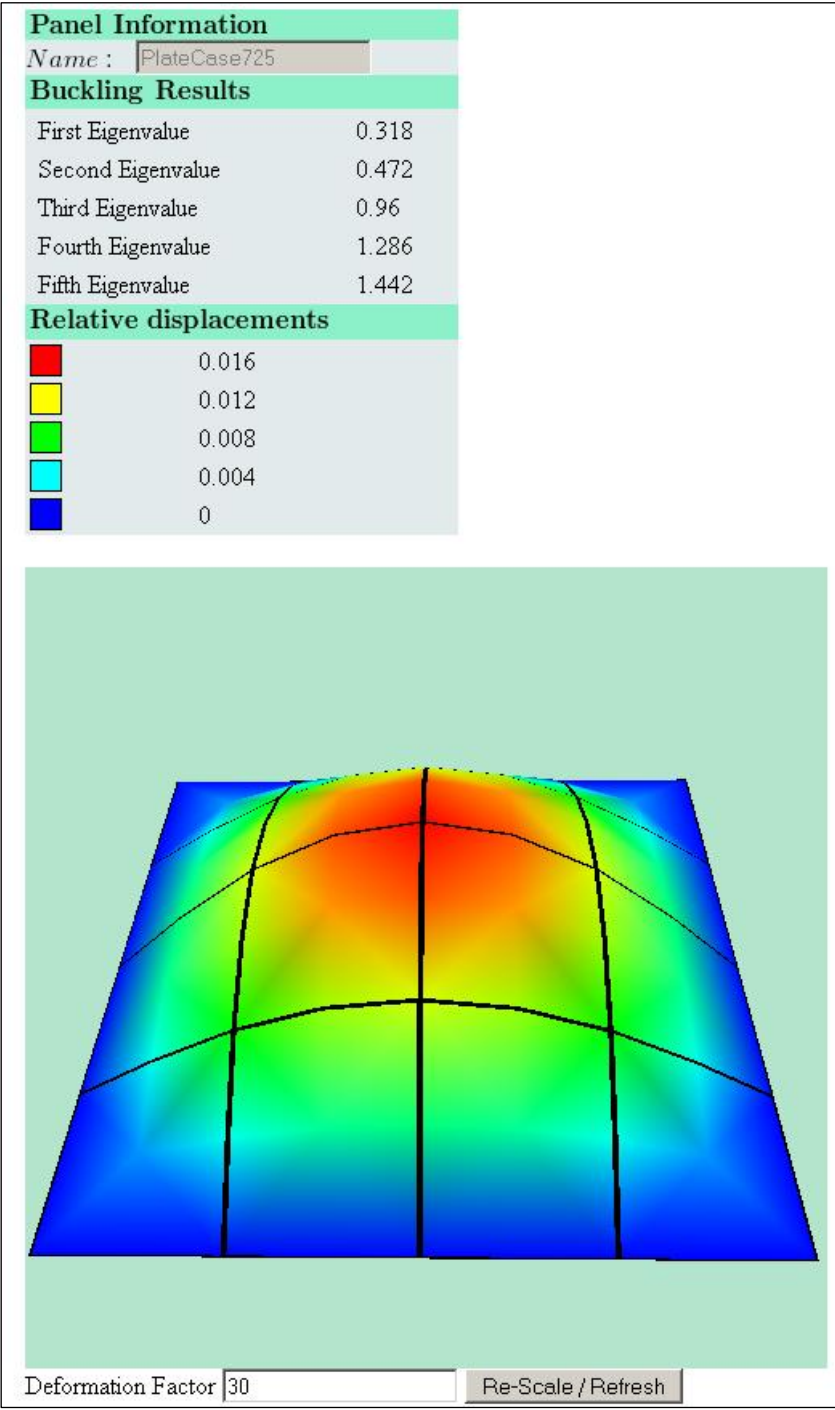


Figure 9.6: Deformed shape of a flat plate with simply supported boundary under biaxial compression using CADEC.

The difference in this case is just 0.12%.

9.2. Validation of a curved panel subject to compression analyzed with ANSYS

The application allows the user to create plates or stiffened panels with a certain curvature. These structural elements are used commonly in the aerospace industry, where the aircraft wings and its fuselage are not made exclusively of flat elements. There are special analyses for flat and cylindrical elements in the literature, but it is not that easy to find methodologies capable of predicting the buckling load of smoothly curved panels. However, as it was explained in Chapter 5, the solver employed by CADEC assumes that the mesh elements are flat, so in order to obtain reasonable results, it is important to highlight that the curvature should be low (moreover considering the coarse mesh used in the application).

In order to validate the results provided by our application, a curved plate made of steel and subject to uniaxial compression will be studied. Results will be compared to those obtained by the commercial finite element analysis program ANSYS®, provided by Le Tran in [31].

9.2.1. Curved panel analyzed with ANSYS

The plate under analysis is shown in the following figure:

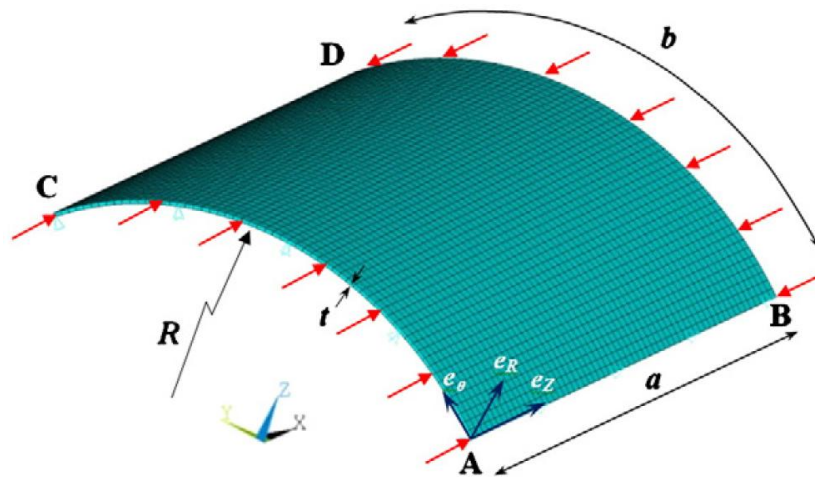


Figure 9.7: Layout of the curved panel analyzed with ANSYS®. Source: Figure 3 of [30].

The plate has dimensions of 1500 x 1500 mm, with a radius of curvature of 4500 mm and thickness of 20 mm. The plate is loaded with a uniformly distributed load along the curved edges. The plate is simply supported on unloaded edges (sides CD and AB), and it is clamped on the loaded edges (sides AC and BD).

The material properties of the steel used in the simulation are $E = 210 \text{ GPa}$ and $\nu = 0.3$. Since steel is an isotropic material, the way of creating this type of material within CADEC will be explained in the next section.

The mesh employed in this simulation is 40 first order quadrangular elements. In order to achieve a similar result with CADEC, it is important to try to get a mesh as similar as possible to the one used by ANSYS®.

The solution obtained when running the previous case with ANSYS® was provided by Le Tran [31] in terms of the stress resulting from loading the plate with the critical buckling load:

$$\sigma^{CR} = 320.9 \text{ MPa}$$

In the next section it will be explained how to model this case with CADEC.

9.2.2. Curved panel analyzed with CADEC

First of all it is necessary to define the material properties of the steel. In order to create isotropic materials within CADEC, it is just necessary to remember its definition (in isotropic materials, material properties are not dependent on the direction). Therefore, $E_1 = E_2$, $\nu_{12} = \nu_{23}$ and $G_{12} = \frac{E_2}{2(1+\nu_{12})}$. The values are summarized in the next table:

Lamina mechanical properties	
E_1 (GPa)	210
E_2 (GPa)	210
ν_{12}	0.3
ν_{23}	0.3
G_{12} (MPa)	80770
Thickness (mm)	20

Table 9.9: Mechanical properties of the lamina used in CADEC for the curved panel case.

Once the lamina and the laminate have been defined (the laminate just contains one steel lamina of 20 mm thickness), we are ready to define the plate. At this point it is important to plan how it is more convenient to model the plate so that the mesh is as similar as possible to the one of ANSYS®. By default, for square plates CADEC employs 4x4 second order quadrangular elements. Thus, 16 elements will be used. Since they are second order elements, the accuracy of the solution should be similar to the one provided by a mesh of 8x8 first order elements. Even though this solution should be similar to the one provided by ANSYS®, a quarter of the plate will be studied in order to refine even more the mesh and try to converge to the real solution. Thus, by analyzing a quarter of the plate using 4x4 second order elements, the accuracy of the results should be similar to the ones obtained simulating the whole plate with a mesh of 8x8 second order elements (about 16x16 first order elements).

The geometric values, BC and load configurations are summarized in the next table:

Geometric Parameters			BC	LOADS	
				Nr	Ns
Curvature	4500	Side 1	CC	1	0
StringersNum	0	Side 2	SS	0	0
Width (A)	750	Side 3	SYM	0	0
Length (B)	750	Side 4	SYM	0	0

Table 9.10: Loads, BC and geometry input for the curved plate in CADEC.

After running the problem with CADEC, the first eigenvalue obtained is:

$$\lambda_1 = 6214.95$$

And from this value it is possible to calculate the critical buckling load per unit length as follows (critical linear pressure):

$$Pressure^{CR} = \lambda_1 \cdot N_r$$

$$Pressure^{CR} = 6214.95 \cdot 1 \text{ N/mm} = 6215 \text{ N/mm}$$

In order to compare the previous result with the literature it is necessary to compute the critical stress:

$$\sigma^{CR} = \frac{Pressure^{CR}}{thickness}$$

$$\sigma^{CR} = \frac{6215 \text{ N/mm}}{20 \text{ mm}} = \mathbf{310.75 \text{ MPa}}$$

On the other hand, the buckling load per unit length obtained with ANSYS® in [31] was:

$$\sigma^{CR} = 320.9 \text{ MPa}$$

Which results in a difference of:

$$\varepsilon = 3.16 \%$$

Which confirms the accuracy of the application. The outcome provided by CADEC is shown in next figure, including the deformation plot of the panel:

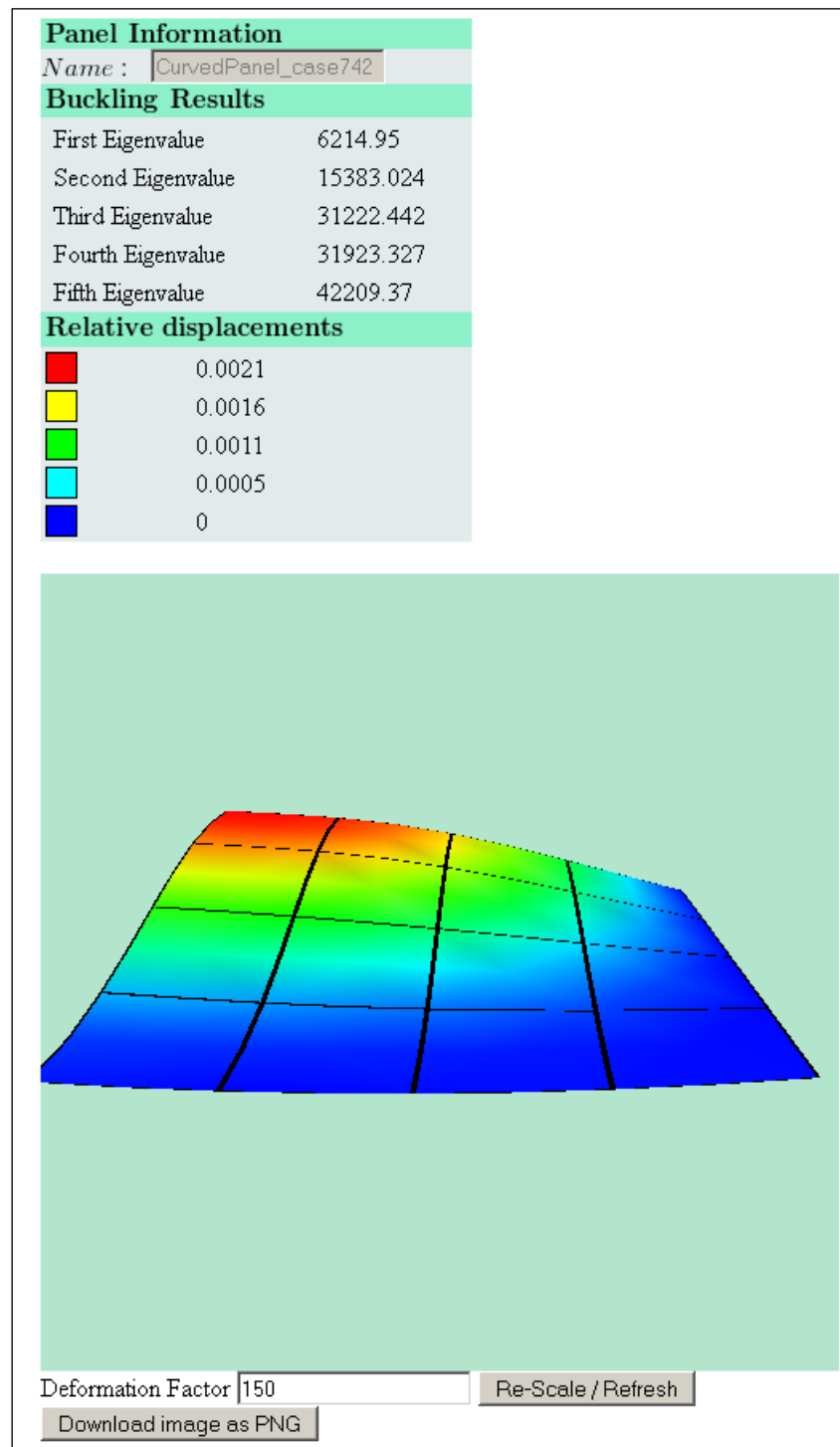


Figure 9.8: Buckling results using CADEC of the curved panel subject to axial compression described in [31], using symmetry.

And the deformation plot of the whole panel would be corresponding to the first buckling mode would be:

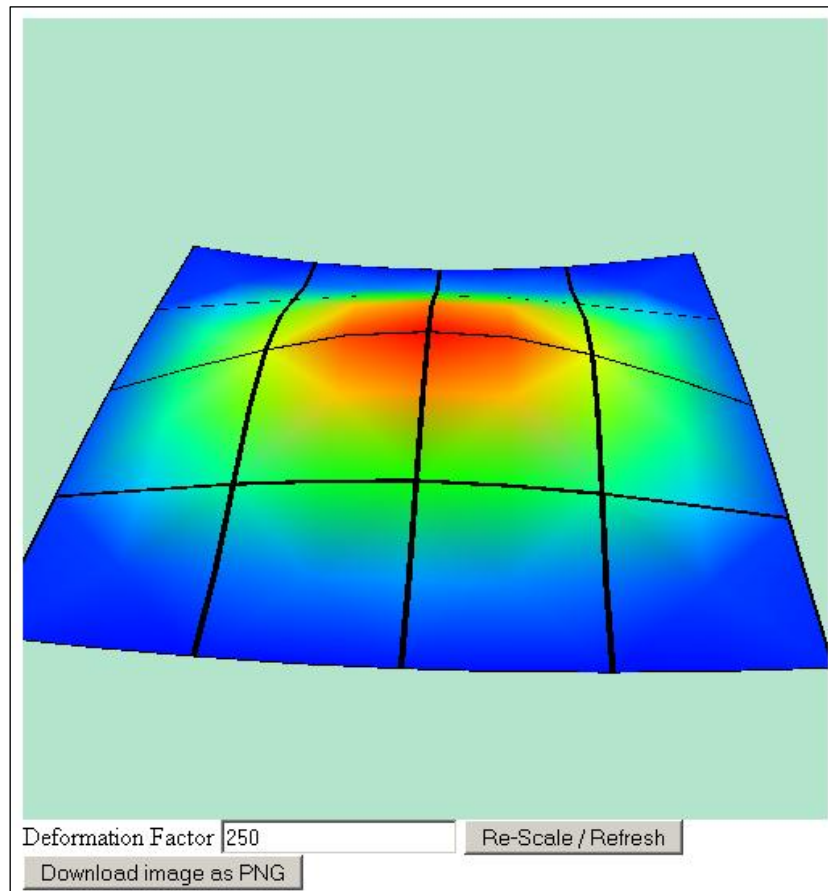


Figure 9.9: Deformed shape of the curved panel subject to axial compression described in [31] (whole model) using CADEC.

9.3. Validation of flat stiffened panels using experimental data

9.3.1. Flat stiffened panel with hat stringers

In order to validate our application with stiffened panels, the first case under consideration is the one presented by Faruk Elaldi in [17, 18]. In this project, buckling and post-buckling behavior of a hat stiffened composite panel is analyzed both using FE models and experimental data. The panel is subject to uniaxial compression in the direction of the stiffeners.

The panel subject to study has the following layout:

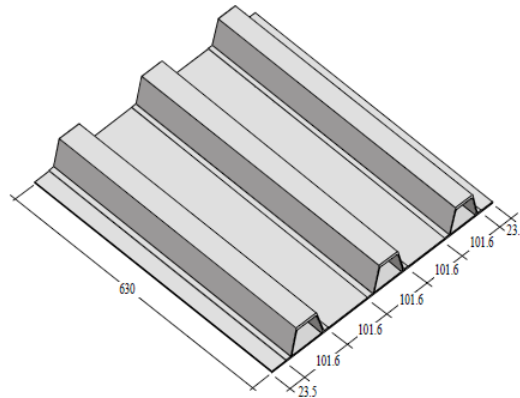


Figure 9.10: Layout of the stiffened panel with three hat stringers. Source: Figure 2 of [17].

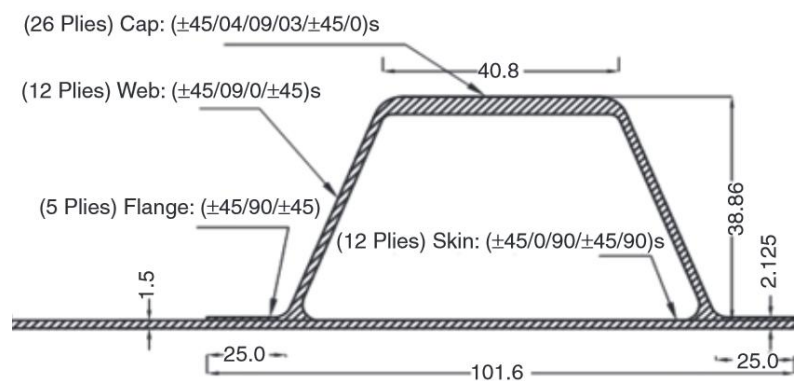


Figure 9.11: Hat stringers profile layout. Source: Figure 1 of [18].

The geometric properties of the panel and stringers are summarized in the next table. Note that some of the values have been rounded for simplicity. In the FE model shell elements are being used, and therefore the distances should be from the mid-surface of each laminate. Then, since thicknesses are not taken into account in the distances, dimensions do not have to be exact (we are already neglecting the thicknesses of the laminates):

Panel dimensions		Stringer geometry	
Curvature	None	Profile	Omega (Hat)
Length(B)	560	Stringer Height	39
Width (A)	554	Stringer Total Width	102
Stringers Number	3	Top Flange Width	41
Pitch	203	Bottom Flange Width	25

Table 9.11: Geometry of the Hat-stiffened panel introduced in CADEC.

Notice that panel total width has been set so that the sub-skins on the free edges have a width of 23 instead of 23.5 mm. Despite the fact that the Length of the panel shown in figure 9.13 is 630 mm, when loading the panel both sides 1 and 3 are clamped, so the effective length after adding the clamping devices is just 560 mm.

The stiffened panel has been manufactured with *Hercules unidirectional IM7 carbon fiber* pre-impregnated with *Cytec's Rigidite 5250-4 resin* (graphite/epoxy). The mechanical properties of such material are shown in the next table:

Lamina material properties			
Lamina mechanical properties		Lamina strength values	
E_{1T} (GPa)	179.5	F_{1T} (GPa)	3.068
E_{1C} (GPa)	138.4	F_{1C} (GPa)	1.684
E_{2T} (GPa)	10.6	F_{2T} (GPa)	0.0482
E_{2C} (GPa)	13.3	F_{2C} (GPa)	0.2576
G_{12} (GPa)	5.0	F_6 (GPa)	0.1382
ν_{12}	0.3		

Table 9.12: Mechanical properties of the lamina for the Hat-stiffened panel.

And the laminates properties can be obtained considering the following stacking sequences:

Skin SS (12 plies)	Bottom Flange SS (5 plies)	Top Flange SS (26 plies)	Web SS (12 plies)
$[\pm 45/0/\pm 45/90]_s$	$[\pm 45/90/\pm 45]$	$[\pm 45/0_4/90/0_3/\pm 45/0]_s$	$[\pm 45/90/0/\pm 45]_s$

Table 9.13: Stacking sequence (SS) of the different laminates forming the Hat-stiffened panel, and their area of application.

The thickness of each ply can be extracted from the figure 9.14 using the thickness of the skin (or the thickness of skin plus bottom flange laminate), resulting in 0.125 mm.

Finally, it is necessary to represent accurately the loading and BC so that they are as similar as possible to the laboratory test conditions. The author explains that the panel was subject to an increasing linearly distributed compressive load in sides 1 and 3 until the ultimate failure of the panel occurred. Then, a unitary value for the load is going to be applied. As explained before, edges 2 and 4 (parallel to stringer's longitudinal direction) are free, and sides 1 and 3 are clamped. It is not clearly specified whether the load is applied just on the skin or both on skin and stringers. However, from the image of the test it seems reasonable to assume that it is loaded just on the skin. These conditions are summarized in the following table:

	BC	LOADS	
		Nr	Ns
Side 1	CC	1	0
Side 2	FREE	0	0
Side 3	CC	1	0
Side 4	FREE	0	0

Table 9.14: Load and BC applied on each side of the Hat-stiffened panel.

After running the problem with CADEC, the first eigenvalue obtained is:

$$\lambda_1 = 170.03$$

And from this value it is possible to calculate the critical buckling load as follows:

$$P^{CR} = \lambda_1 \cdot N_r \cdot L_{applied\ load}$$

Where:

λ_1 : first eigenvalue.

N_r : load per unit length applied to the panel.

$L_{applied\ load}$: total length where N_r is applied. (in this case is the length of sides 1 or 3)

Substituting in the previous expression, the critical buckling load results:

$$P^{CR} = 170.03 \cdot 1\ N/mm \cdot 554\ mm = \mathbf{94.2\ kN}$$

On the other hand, the buckling load obtained from the experimental test in [17] was:

$$P^{CR} = 94.3\ kN$$

Which results in a difference of:

$$\varepsilon = 0.11 \%$$

Proving the accuracy of the application. Moreover, the author mentions in [17] that, when subject to that load, the **skin bay (intermediate skins) buckled into five half-wave lengths**, which is the exact buckling mode predicted by our application. The outcome provided by CADEC is shown in next figure, including the deformation plot of the panel.

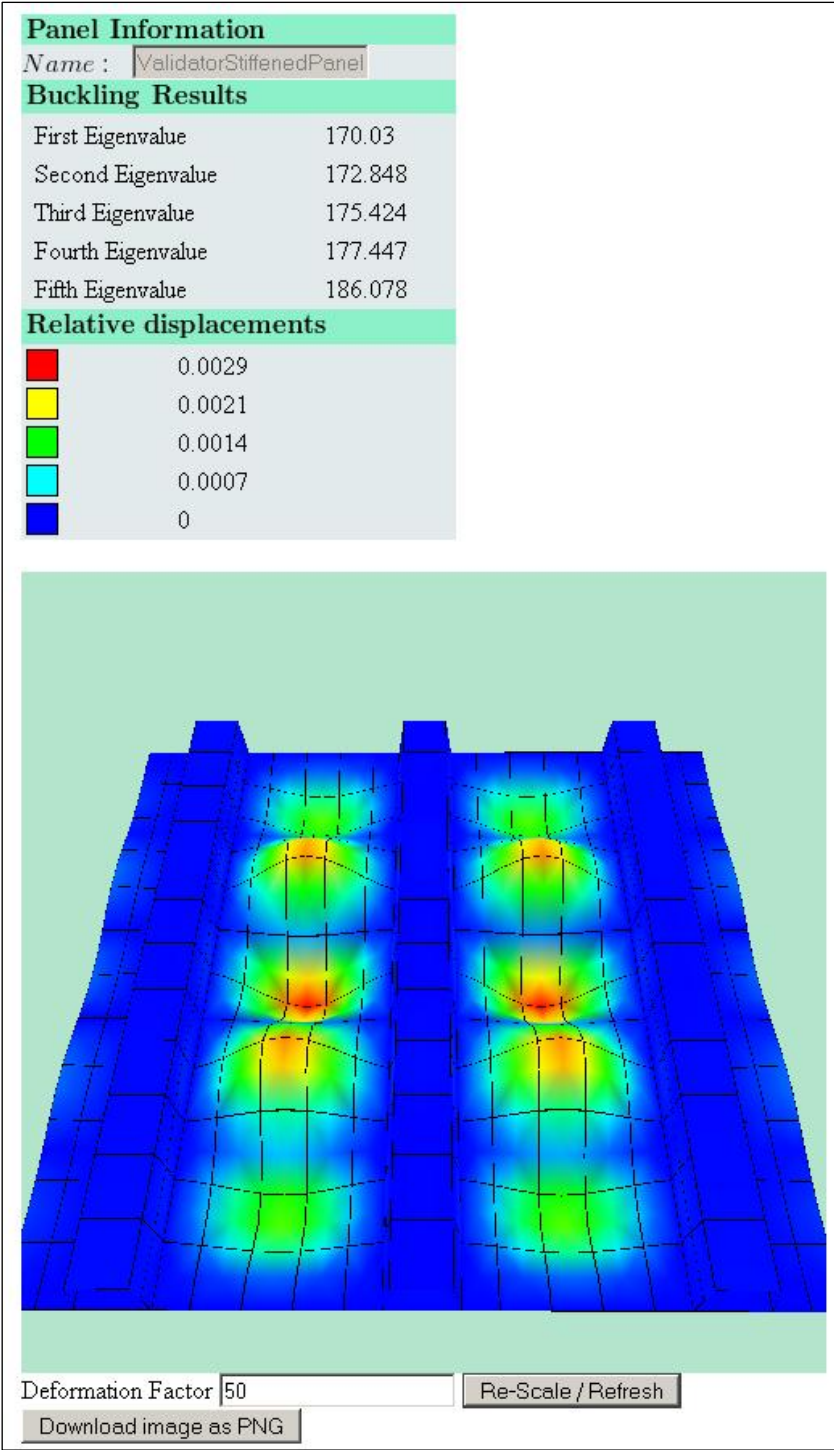


Figure 9.12: Buckling results of the stiffened panel with three hat stringers using CADEC.

9.3.2. Flat stiffened panel with JC stringers

The second case that will be studied is also presented by Faruk Elaldi in [18]. In this project, buckling and post-buckling behavior of a Z stiffened composite panel is analyzed both using FE models and experimental data. The panel is subject to uniaxial compression in the direction of the stiffeners.

The panel subject to study has the following layout:

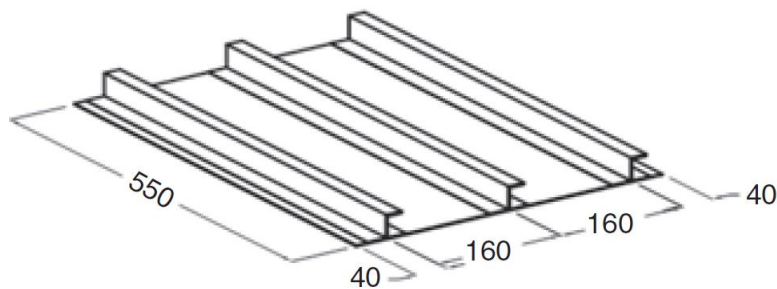


Figure 9.13: Layout of the stiffened panel with three JC stringers. Source: Figure 2 of [18].

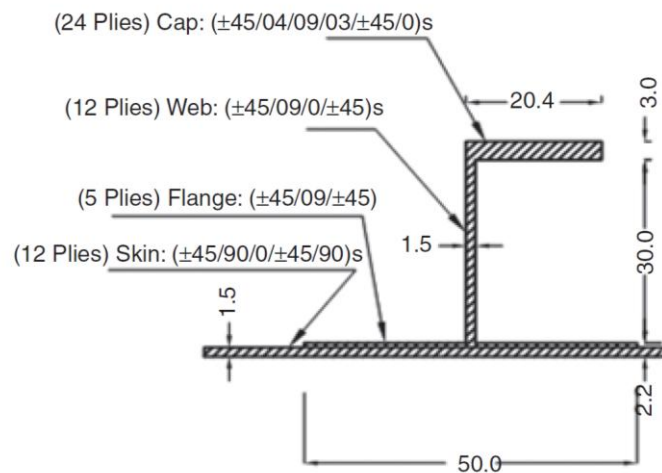


Figure 9.14: JC stringers profile layout. Source: Figure 1 of [18].

The geometric properties of the panel and stringers are summarized in the next table. Note that some of the values have been rounded for simplicity as in the previous case:

Panel dimensions		Stringer geometry	
Curvature	None	Profile	JC
Length(B)	480	Stringer Height	30
Width (A)	450	Stringer Total Width	50
Stringers Number	3	Top Flange Width	-
Pitch	160	Bottom Flange Width	-

Table 9.15: Geometry of the JC-stiffened panel introduced in CADEC.

Notice that panel's total width has been set so that the sub-skins on the free edges have a width of 40 mm. Despite the fact that the Length of the panel shown in figure 9.16 is 550mm, when loading the panel both sides 1 and 3 are clamped, so the effective length after adding the clamping devices is just 480 mm. Furthermore, our application contains a definition of "JC" profile in which top flange width is computed to have the same width as each of the bottom flanges, so in this case it would be 25 mm wide instead of the 20.4 mm shown in the figure. However, results should not be affected importantly by this small correction.

The stiffened panel has been manufactured with *Hercules unidirectional IM7 carbon fiber* pre-impregnated with *Cytec's Rigidite 5250-4 resin* (graphite/epoxy). The mechanical properties of such material are shown in the next table:

Lamina material properties			
Lamina mechanical properties		Lamina strength values	
E_{1T} (GPa)	179.5	F_{1T} (GPa)	3.068
E_{1C} (GPa)	138.4	F_{1C} (GPa)	1.684
E_{2T} (GPa)	10.6	F_{2T} (GPa)	0.0482
E_{2C} (GPa)	13.3	F_{2C} (GPa)	0.2576
G_{12} (GPa)	5.0	F_6 (GPa)	0.1382
ν_{12}	0.3		

Table 9.16: Mechanical properties of the lamina for the JC-stiffened panel.

And the laminate properties can be obtained considering the following stacking sequence:

Skin SS (12 plies)	Bottom Flange SS (5 plies)	Top Flange SS (26 plies)	Web SS (12 plies)
$[\pm 45/0/\pm 45/90]_s$	$[\pm 45/90/\pm 45]$	$[\pm 45/0_4/90/0_3/\pm 45/0]_s$	$[\pm 45/90/0/\pm 45]_s$

Table 9.17: Stacking sequence (SS) of the different laminates forming the JC-stiffened panel, and their area of application.

The thickness of each ply can be extracted from the figure 9.17 using the thickness of the skin (or the thickness of skin plus bottom flange laminate), resulting in 0.125 mm.

Finally, it is necessary to represent accurately the loading and BC so that they are as similar as possible to the laboratory test conditions. The author explains that the panel was subject to an increasing linearly distributed compressive load in sides 1 and 3 until the ultimate failure of the panel occurred. Then, a unitary value for the load is going to be applied. As explained before, edges 2 and 4 (parallel to stringer's longitudinal direction) are free, and sides 1 and 3 are clamped. These conditions are summarized in the following table:

	BC	LOADS	
		Nr	Ns
Side 1	CC	1	0
Side 2	FREE	0	0
Side 3	CC	1	0
Side 4	FREE	0	0

Table 9.18: Load and BC applied on each side of the JC-stiffened panel.

After running the problem with CADEC, the first eigenvalue obtained is:

$$\lambda_1 = 132.39$$

And from this value it is possible to calculate the critical buckling load as follows:

$$P^{CR} = \lambda_1 \cdot N_r \cdot L_{applied\ load}$$

$$P^{CR} = 132.39 \cdot 1\ N/mm \cdot 450\ mm = \mathbf{59.58\ kN}$$

On the other hand, the buckling load obtained from the experimental test in [18] was:

$$P^{CR} = 58.1\ kN$$

Which results in a difference of:

$$\varepsilon = 2.54\ \%$$

Again, the application was capable of predicting the buckling load with great accuracy. The outcome provided by CADEC is shown in next figure, including the deformation plot of the panel.

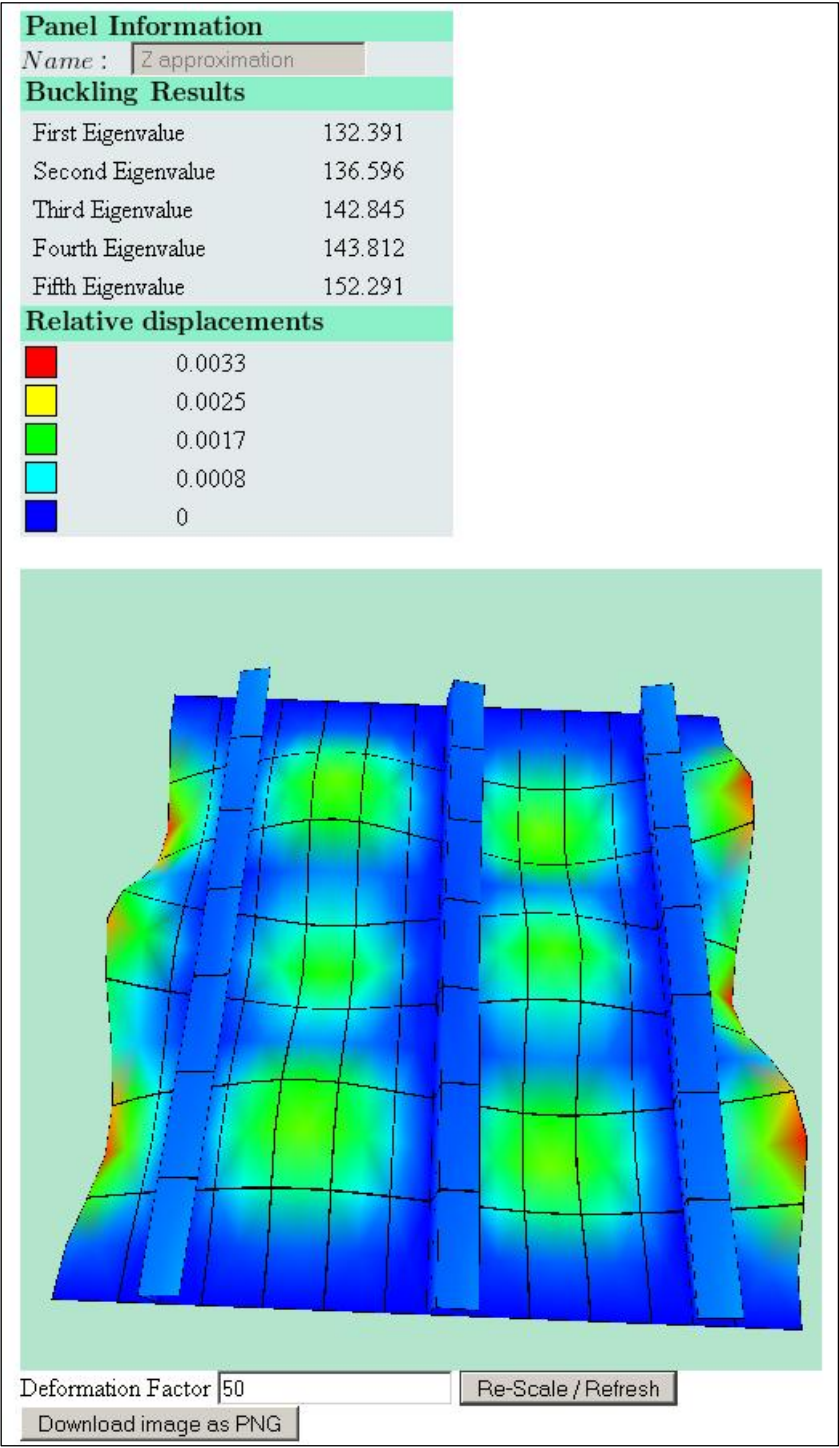


Figure 9.15: Buckling results of the stiffened panel with three JC stringers using CADEC.

9.3.3. Flat stiffened panels with J stringers

The last case that will be studied is presented in [11]. In this project five stiffened panels with three different geometries are tested in compression beyond initial buckling to collapse. Due to the incapability of reproducing the exact geometry of the experiment with CADEC, larger errors are expected than in the previous cases. However, the approximated models will be similar enough to expect reasonable results.

A. Experimental results

The panels subject to study have the following profiles:

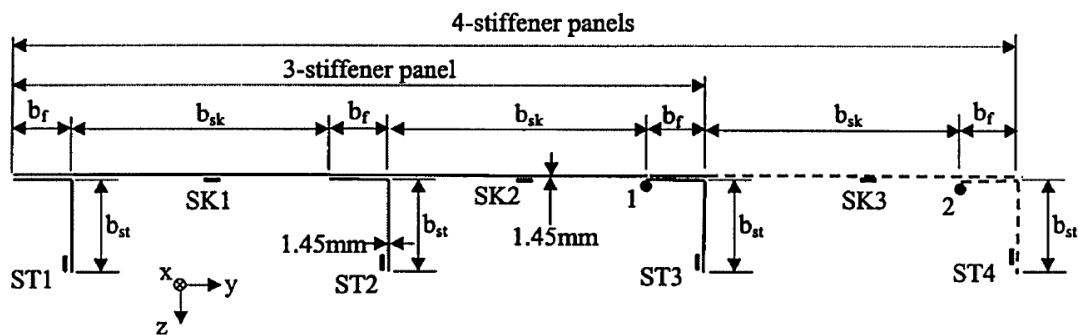


Figure 9.16: Layout of the three different configurations of stiffened panel using J stringers. Source: Figure 2 in [11].

The previous image shows the layout of the three different configurations. First panel has 3 stringers, and the other two have four. As it can be seen in the figure, the different between the last two is mainly the orientation of the J stiffeners. The symmetric panel can just be simulated by CADEC using symmetry boundary condition and studying half model.

The geometric properties of the experimental specimens are summarized in the following table, which are slightly different from the ones introduced in CADEC, as it will be explained later.

Geometric properties (Real dimensions)			
Case 1: 3J		Case 2: 4J-Asym	Case 3: 4J-Sym
Width A (mm)	300	260	256
Length B (mm)	500	400	410
Num of stringers	3	4	4
b_{st} (mm)	35	25	23
b_{sk} (mm)	122.25	62	72
b_f (mm)	18.5	18.5	10
Pitch (mm)	140.75	80.5	82

Table 9.19: Geometry of the three experimental different J-stiffened panels.

The material used both for the skin and the stringers is Aluminum, and the thickness is of 1.45 mm both for stringers and skin. They were manufactured separately with a tolerance in all dimensions but thickness of ± 1 mm, and then they were joined using *Loctite Multi-bond 329* adhesive and *7386 Activator*, with rivets to position and secure the stiffener during curing. Material properties are summarized in the following table:

Aluminum stress-strain curve		
σ (MPa)	ε (strain)	E (GPa)
0	0	75
70	1000	70
137	2000	64
200	3000	62
207	3250	52
227	3500	47
248	4000	30

Table 9.20: Mechanical properties of Aluminum used in the three J-stiffened panels.

Since it is not possible to represent the non linearity of the material within CADEC, a value of $E = 75$ GPa will be considered, and a Poisson ratio of $\nu = 0.3$.

Finally, the compression of the panel is carried out by the use of a pair of steel mounts of semi-circular cross-section, placed at either end of the panel to represent the simply-supported end condition. The centre of rotation of these mounts was aligned to coincide with the neutral axis of each panel using a number of positioning bolts. The other two edges are free.

The results provided by [11] after carrying out the compression tests are summarized in the next table:

Buckling Critical Loads			
Case 1: 3J		Case 2: 4J-Asym	Case 3: 4J-Sym
Local Buckling (kN)	29	64	58
Collapse (kN)	55	76	76
Failure Mode	Skin/Overall	Adhesive Failure	Skin/Overall

Table 9.21: Experimental values for the critical buckling load obtained by [11].

Now it is time to model the previous panels using CADEC.

B. Results of CADEC simulation

One of the main problems when attempting to create the previous geometries with CADEC is that the current application does not allow to have no lateral skins. Therefore, lateral skin must be added, and it should be small enough to alter the results as less as possible. In order to prevent the formation of elements with excessive aspect ratio, a lateral panel width of 8 mm for case 1 and of 4 mm for cases 2 and 3 will be used. Furthermore, case three has variable pitch and different orientation of the stringers, so in order to provide similar results symmetry conditions must be applied, and therefore just half of the model will be studied. Furthermore an offset must be applied to move the stringers to the left and achieve a distance from the second stringer to side 2 equal to half the central pitch (half of b_{sk} value). The final geometric properties introduced to our application are gathered in the following table.

Geometric properties			
Case 1: 3J		Case 2: 4J-Asym	Case 3: 4J-Sym (half model)
Width A (mm)	316	268	132
Length B (mm)	500	400	410
Radius (mm)	-	-	-
Num of stringers	3	4	2
Pitch (mm)	140.75	80.5	82
Offset (mm)	0	0	-16
Stringer height (mm)	35	25	23
Stringer width (mm)	18.5	18.5	10

Table 9.22: Geometry of the three JC-stiffened panels introduced in CADEC.

Given that Aluminum is an isotropic material, the following assumptions apply $E_1 = E_2$, $\nu_{12} = \nu_{23}$ and $G_{12} = \frac{E_2}{2(1+\nu_{12})}$. The values introduced to CADEC are summarized in the next table:

Lamina mechanical properties	
E_1 (GPa)	75
E_2 (GPa)	75
ν_{12}	0.3
ν_{23}	0.3
G_{12} (MPa)	28846
Thickness (mm)	1.45

Table 9.23: Mechanical properties of the lamina for the JC-stiffened panels modeled with CADEC.

Finally, it is necessary to represent accurately the loading and BC so that they are as similar as possible to the laboratory test conditions. The author explains that the panel was subject to an increasing linearly distributed compressive load in sides 1 and 3 until the ultimate failure of the panel occurred. Then, a unitary value for the load is going to be applied. As explained before, edges 2 and 4 (parallel to stringer's longitudinal direction) are free, and sides 1 and 3 are simply supported. These conditions are summarized in the following table:

	Case 1: 3J			Case 2: 4J-Asym			Case 3: 4J-Sym (half model)		
	BC	LOADS		BC	LOADS		BC	LOADS	
		Nr	Ns		Nr	Ns		Nr	Ns
Side 1	SS	1	0	SS	1	0	SS	1	0
Side 2	FREE	0	0	FREE	0	0	SYM	0	0
Side 3	SS	1	0	SS	1	0	SS	1	0
Side 4	FREE	0	0	FREE	0	0	FREE	0	0

Table 9.24: Load and BC applied on each side of the three different J-stiffened panels.

After running the problem with CADEC, the first eigenvalue is obtained for each panel. Then, the critical load is computed as in the previous cases by multiplying the eigenvalue times the applied load and the length of the loaded edges. In the third case, the length must be the one of the total panel, not the loaded length of the half model. All these values and the difference in results are summarized in the next table.

Buckling Critical Loads			
Case 1: 3J		Case 2: 4J-Asym	Case 3: 4J-Sym
λ_1	78.1	284.4	231.3
$L_{applied\ load}\ (mm)$	316	268	$132 \times 2 = 264$
$P^{CR} = \lambda_1 \cdot N_r \cdot L_{applied\ load}\ (kN)\ [2]$	24.7	76.2	61
$P^{CR}_{experimental\ data}\ (kN)\ [11]$	28	64	58
<i>difference (%)</i>	11.8 %	19.1%	5.2 %

Table 9.25: Comparisson between the experimental buckling loads [11] and the results provided by CADEC for the three different panels.

The outcome provided by CADEC for the three panels is shown in next figures, including the deformation plots of the panel.

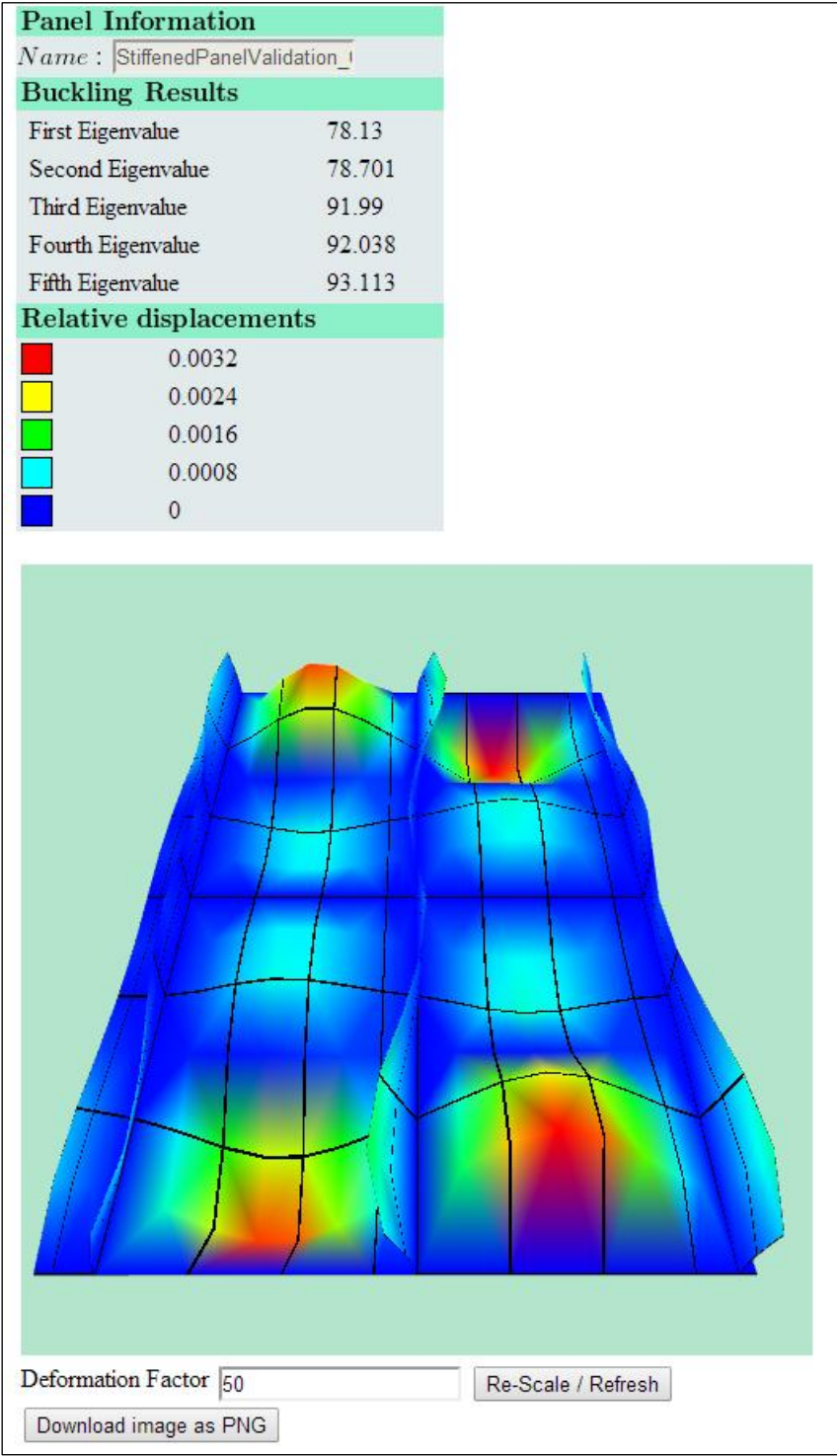


Figure 9.17: Buckling results of case 1: stiffened panel with three J stringers using CADEC.

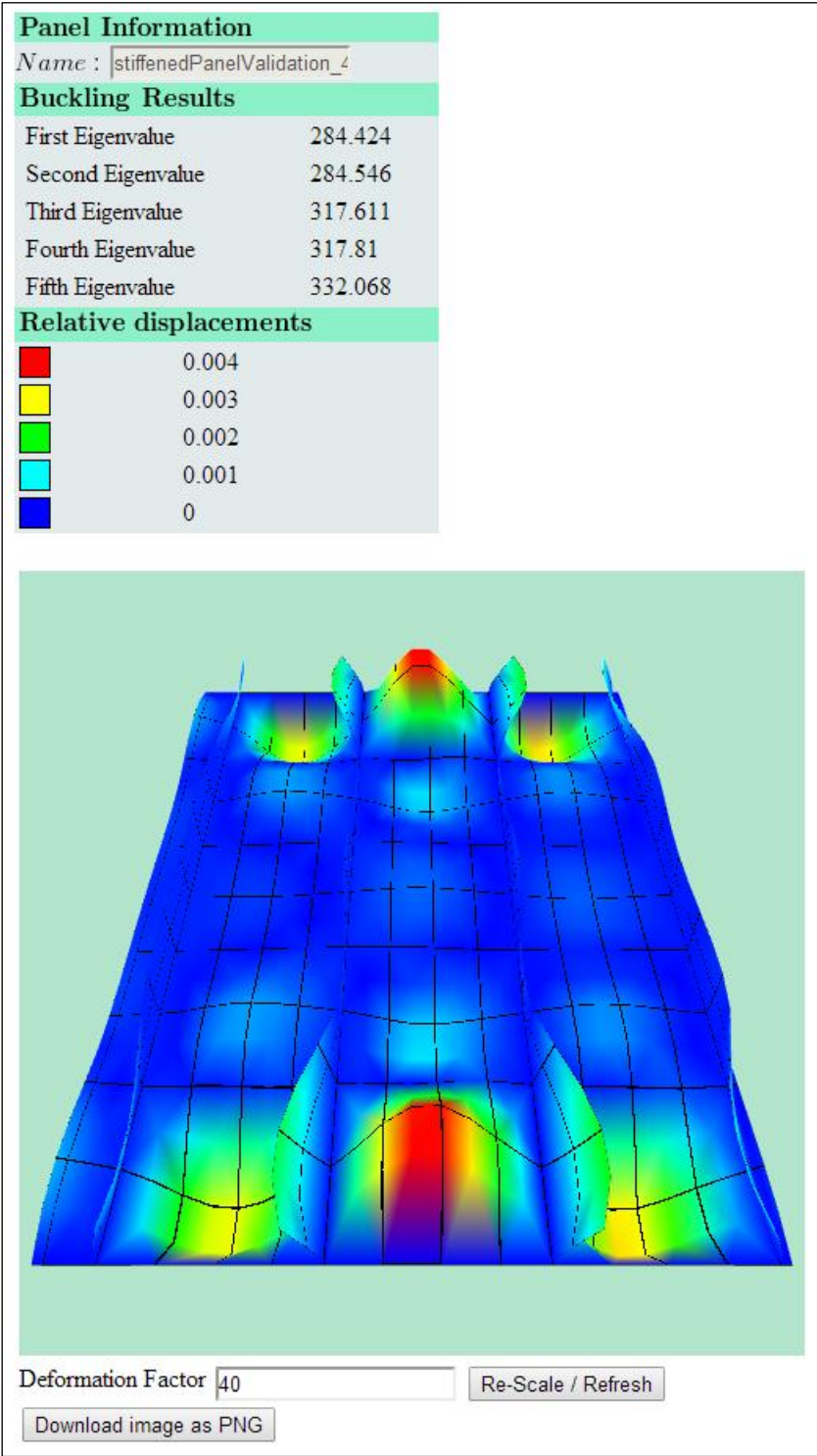


Figure 9.18: Buckling results of case 2: asymmetric stiffened panel with four J stringers using CADEC.

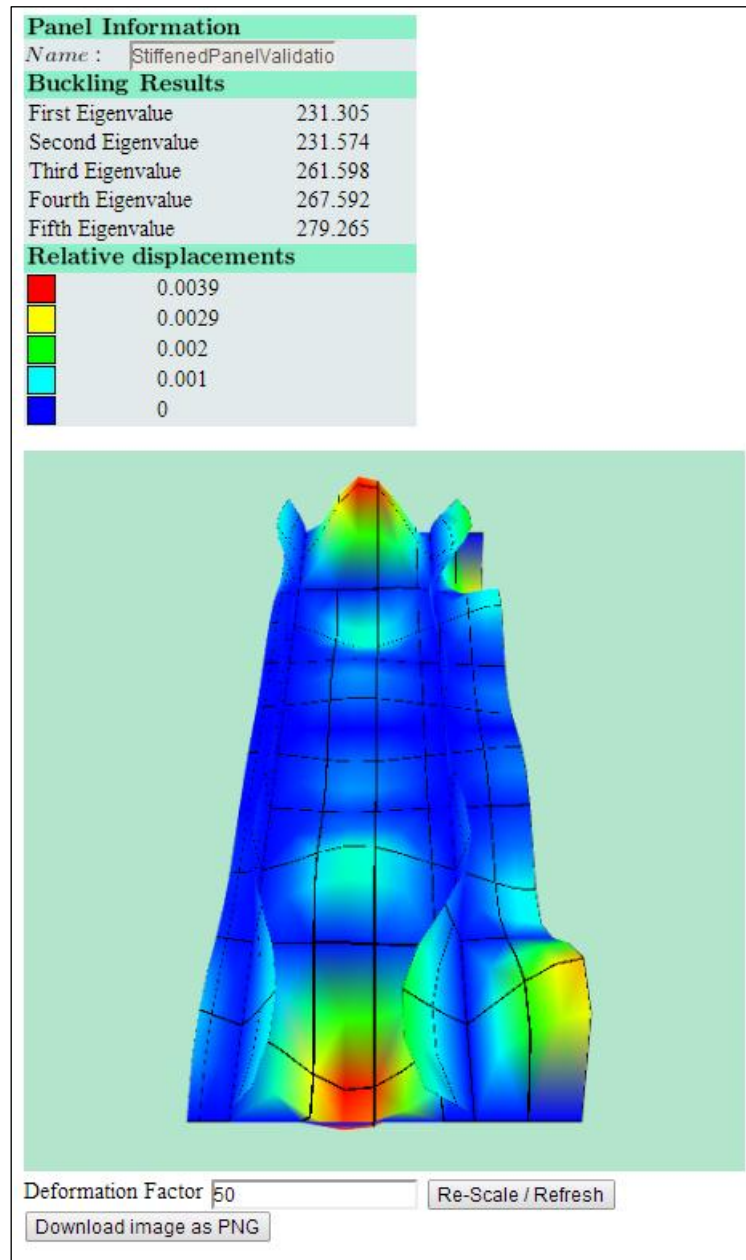


Figure 9.19: Buckling results of case 3: symmetric stiffened panel with three J stringers (half model) using CADEC.

The differences between the results provided by CADEC and the experimental data can be justified by several reasons. First, the geometry used to model the specimens was not exact but an approximation. Additionally, it is being assumed that the material buckles in the elastic linear range (where the E modulus remains constant), but could not be the real situation. Besides, since only one specimen was tested for cases 2 and 3 in the literature, there could be some externalities affecting the experimental results (friction on the loaded edges, eccentricity, humidity, temperature, etc). It would be more accurate to have several specimens tested for each case in order to reduce the uncertainty of the experimental results.

Finally, it is important to understand that the application studies the case of a perfect panel, free of any defect or imperfection. This is not true with real panels, which can contain material imperfections as well as geometric defects to which stability analyses are very sensitive. For example, in the case 2 the load predicted by CADEC was relatively higher than the initial buckling load observed in the test, becoming actually almost equal to the load causing collapse. The reason of that behavior could be explained by the next figure. CADEC will provide the buckling load at the primary bifurcation, which would be the local buckling load of the panel in case it was defect free and ideal. However, the real panel is likely to have some manufacturing defects that will lead it to follow the imperfect path. Due to those imperfections, the actual load at which first evidence of buckling are revealed is lower. Then the panel will absorb some more load along the secondary imperfect path until failure. In the figure, the red dot represents CADEC critical buckling load, whereas the orange dot stands for the real buckling load of the specimen.

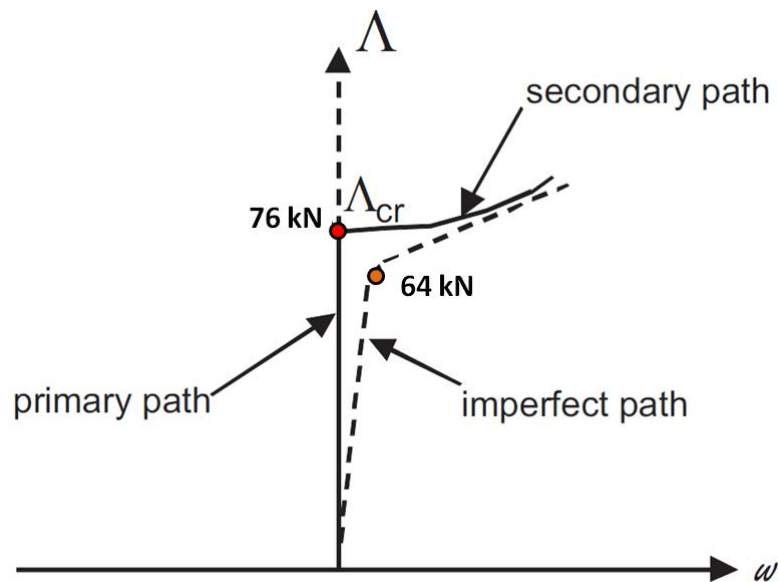


Figure 9.20: Buckling critical load for ideal and imperfect paths. Source: Edition of Image 4.1 in [5].

No matter the exact reasons behind the differences in results, it is relevant to highlight that the application provides a good approximation of the real buckling loads. Since its main use will be at the pre-design step, differences below 20% are accepted, even more when in the rest of the cases analyzed such differences were much lower than 10%.

9.4. Validation of curved stiffened panel analyzed with Abaqus

9.4.1. Curved T-stiffened panel analyzed with Abaqus

It would have been interesting to validate the results provided by CADEC for a curved stiffened panel by means of experimental data, but it has not been possible to find trustful resources for such data. Most of the cases found in the literature did not include all the parameters necessary to define the problem. Therefore, a FE model will be defined with Abaqus® to validate this type of problems.

The panel subject to study will be a curved integrally-stiffened panel containing four “T” stringers. The geometric properties are summarized in the next table:

Panel dimensions		Stringer geometry	
Curvature	3000	Profile	T
Length(B)	800	Stringer Height	40
Width (A)	600	Stringer Total Width	40
Stringers Number	4	Top Flange Width	-
Pitch	150	Bottom Flange Width	-

Table 9.26: Geometry of the curved T-stiffened panel defined in Abaqus®.

The stiffened panel will be made of aluminum, with properties defined in table 9.20, and thickness 1.45 mm. Material properties are applied by creating a general stiffness shell section and introducing the ABDH matrices of that laminate.

The panel is meshed applying an approximate element size of 20 mm, which results in 1600 elements. Curved edges are loaded applying a normal shell load of 1 N/mm, whereas the straight edges are free. Loaded sides are considered clamped, and therefore the following DOFs are restrained in those edges (CSYS where Y axis is normal to the skin):

$$u_y = 0$$

$$\varphi_x = 0$$

$$\varphi_z = 0$$

In order to avoid rigid solid movements, the DOF u_x is restrained in two corner nodes so that lateral vibration is prevented.

After running the model with Abaqus®, the first eigenvalue obtained is:

$$\lambda_1 = 41.28$$

And from this value it is possible to calculate the critical buckling load as follows:

$$P^{CR} = \lambda_1 \cdot N_r \cdot L_{applied\ load}$$

$$P^{CR} = 41.28 \cdot 1\ N/mm \cdot 600\ mm = 24.77\ kN$$

The deformation plot of the panel is shown in next figure:

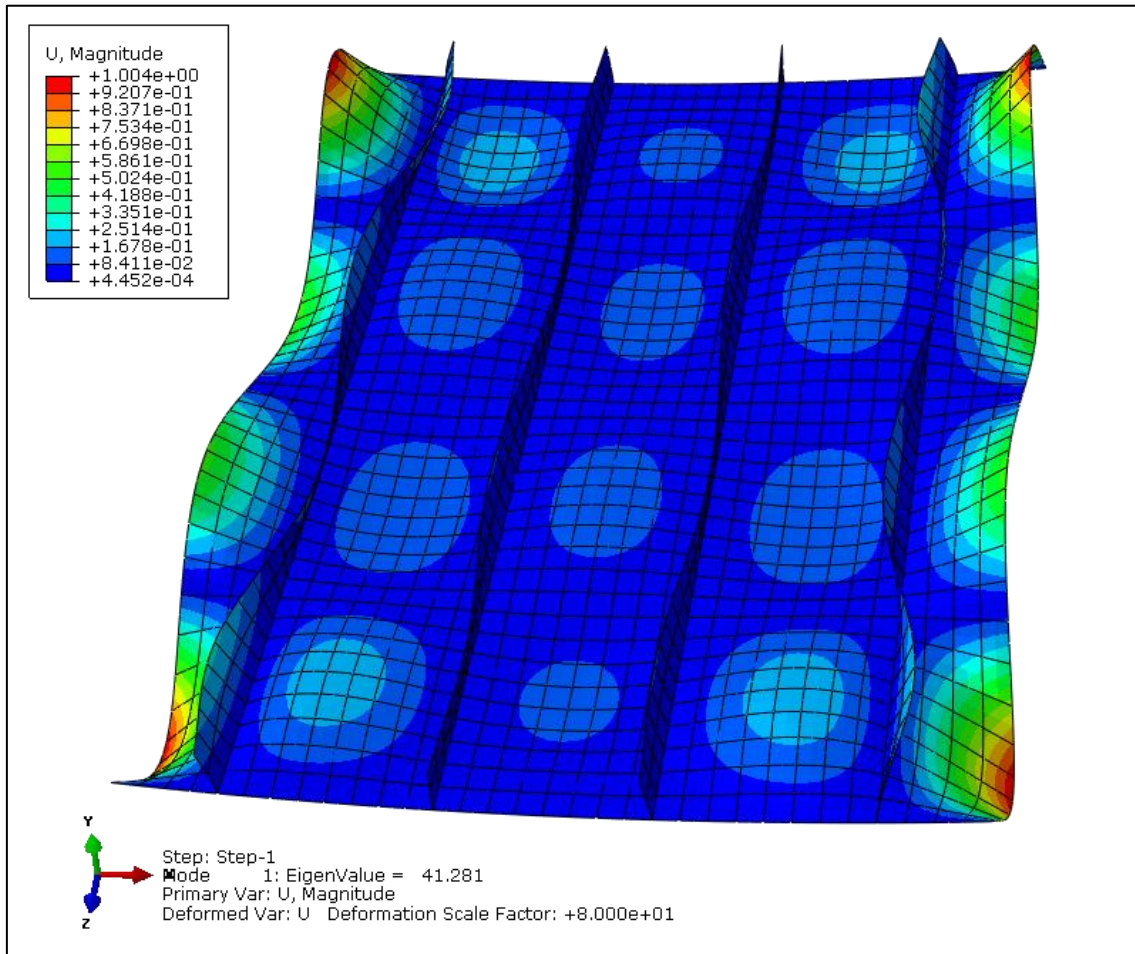


Figure 9.21: Buckling results of a curved stiffened panel with four T stringers using Abaqus®.

9.4.1. Curved T-stiffened panel analyzed with CADEC

The same geometry modeled with Abaqus® is created in CADEC by introducing the values summarized in the next table:

Panel dimensions		Stringer geometry	
Curvature	3000	Profile	T
Length(B)	800	Stringer Height	40
Width (A)	600	Stringer Total Width	40
Stringers Number	4	Top Flange Width	-
Pitch	150	Bottom Flange Width	-

Table 9.27: Geometry of the curved T-stiffened panel introduced in CADEC.

The stiffened panel will be made of aluminum, with properties defined in table 9.20, and thickness 1.45 mm. Since the panel is integrally-stiffened and it is not possible to introduce zero thickness for a laminate, bottom flange laminates will have negligible thickness: 0.001 mm.

The panel is meshed automatically by CADEC, resulting in 435 elements. BC and load are applied as detailed in next table to achieve same conditions as simulated with Abaqus®:

	BC	LOADS	
		Nr	Ns
Side 1	CC	1	0
Side 2	FREE	0	0
Side 3	CC	1	0
Side 4	FREE	0	0

Table 9.28: Load and BC applied on each side of the curved T-stiffened panel.

After running the problem with CADEC, the first eigenvalue obtained is:

$$\lambda_1 = 42.41$$

Being possible to compute the critical buckling load as follows:

$$P^{CR} = \lambda_1 \cdot N_r \cdot L_{applied\ load}$$

$$P^{CR} = 42.41 \cdot 1 \text{ N/mm} \cdot 600 \text{ mm} = \mathbf{25.45 \text{ kN}}$$

On the other hand, the buckling load obtained with Abaqus® was:

$$P^{CR} = 24.77 \text{ kN}$$

Which results in a difference of:

$$\varepsilon = 2.75 \%$$

Despite using a mesh four times coarser, the application was capable of predicting the buckling load with great accuracy. The deformation plot provided with Abaqus® and CADEC are the same, as shown in next figure:

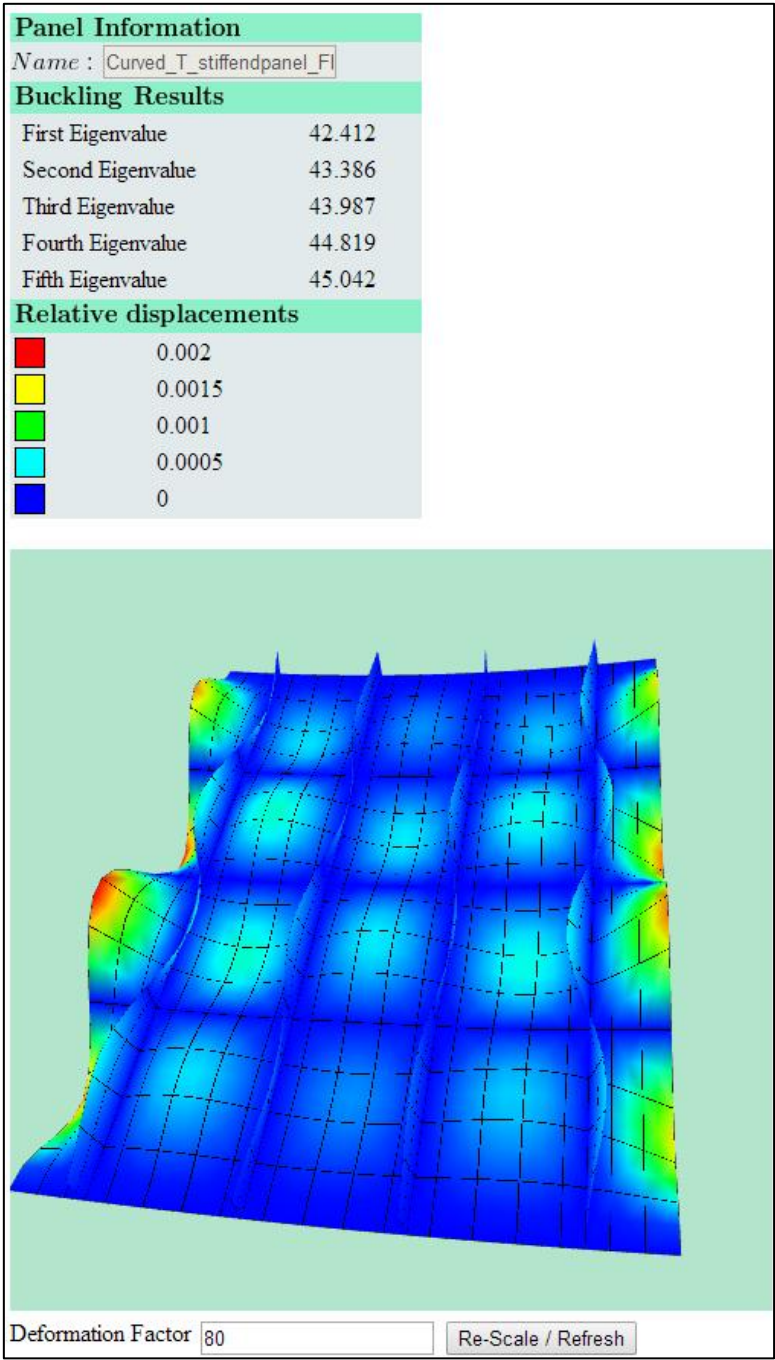


Figure 9.22: Buckling results of a curved stiffened panel with four T stringers using CADEC.

Chapter 10

10. Conclusions and future work

10.1. Conclusions

In this project, a new application serving the design of composite stiffened panels was designed and implemented in a web environment. The application is capable of generating the geometry and mesh for multiple configurations of stiffened panels from a few input parameters, and by performing Finite Element Analysis provides the user with the buckling load and the deformed shape of such panel. Important improvements in comparison to SPD have been achieved. The application has accelerated and simplified the overall design process, revealing the following observations:

- Since stiffened panels are thin-walled structures, the main analysis carried out during the design step is a buckling analysis. Buckling emerges when the structure is subject to in-plane compression or shear loading. CADEC is now capable of performing such analysis in an intuitive way, obtaining the buckling load of the first five modes as well as a representation of the deformed structure once that critical load is reached. CADEC resolution of shear loaded panels is prepared to be available in a near future version, as soon as the solver is updated.

- The application allows a wide range of possible panel configurations. Users may analyze plates, skin-stringer panels, and integrally-stiffened panels. Flat or slightly curved panels may be chosen, though cylindrical profiles are not allowed.
- One of the main problems regarding FEA is the influence of mesh density. Finer mesh provides more accurate results, but increases time required to solve the problem. Since CADEC is a web application whose server must solve several analyses simultaneously and rapidly, time consumption took on high importance. After running hundreds of different cases, the mesh generator tool was configured to ensure accurate results as rapidly as possible. The current version of CADEC has limited the resolution time to two minutes. Depending on the geometry of the panel, the tool is capable of increasing or decreasing mesh density to stay beneath that boundary. However, there are a limited number of stringers that a panel can contain given that time limit.
- To speed up the design process, some of the parameters are simplified. For example, pitch is assumed to be constant, and in some profiles bottom and top flanges must have the same width. This simplification does not prevent the user from obtaining good results using a simplified model. In the design stage it is very common to perform these kinds of assumptions for simplicity, while real geometry is analyzed more accurately after a satisfactory preliminary design of the panel. However, due to offset and the symmetric boundary condition, it is possible to represent some of these cases with CADEC, as in some of the cases analyzed in the chapter on validation.
- Though the application was conceived to solve panels made of composites due to their widespread use in industry and the special challenge to their analysis using traditional formulations, it can also analyze isotropic materials such as steel or aluminum.
- Multiple comparisons to experimental or analytical results were carried out, covering most possible cases such as flat or curved plates, several loading and support conditions, and stiffened panels with different stringer profiles. In most cases satisfactory results of design discrepancies under 20 % are allowed were achieved.
- Since the problem is reduced to a few input parameters, it is possible to perform modifications quickly without developing a new FE model for each modification. This is especially valuable in the preliminary design stage when most of the geometrical changes take place. This feature represents a real advantage over general purpose FE commercial software.
- The implementation of the application in a web environment allows the user to work on the design from any location or web-enabled device, and the database stores all data for a later use. This flexibility increases the application's accessibility.

- The visualization tool implemented in the website provides extensive information. The tool can zoom in or out, rotate, or move the panel to analyze every detail of the deformed structure. The color range used highlights the areas where larger displacements take place. These visual representations implanted within the web application present information more rapidly than any external program.

10.2. Possible future work

Based on the results achieved in this project, several lines of future work to improve the application follow:

- Despite the fact that most of these structures are subject to in-plane loads, it may be interesting to analyze cases in which out-of-plane loads are applied.
- Some cases could be run under shear loading and compared with experimental data for further validation, as soon as the solver is updated.
- Code could be improved so that Symmetric BC cannot be applied on the panel if shear loads are applied, and a new asymmetric BC defined for these cases.
- As BMI3 computes the eigenvectors for the modes 2 to 5 (displacements of the deformed panel nodes), it could be interesting to represent the displacements of these modes in the visualization page.
- BMI3 is capable of computing additional information about the secondary path, including whether it is stable or unstable, and the approximate slope. It would be interesting to provide the user with this information about whether the panel collapses once the critical load is reached or whether it can deal with more load.
- Verification could confirm whether the composite material fails before reaching the critical buckling load. At present, buckling load is obtained based on material properties, panel geometry, and applied load and boundary conditions, but no material failure criterion is used.

Bibliography

- [1] *An introduction to WebGL* [homepage on the Internet]. [updated 13 October 2011, accessed 28 January 2014]. Available from: <http://dev.opera.com/articles/view/an-introduction-to-webgl/>
- [2] Barbero E. J., *CADEC (Computer Aided Design Environment for Composites)* [homepage on the Internet]. Morgantown: Mechanical and Aerospace Engineering Department of West Virginia University; 2010 [updated 1 March 2014, cited 12 March 2014]. Available from: www.cadec-online.com.
- [3] Barbero E. J., Dede E., Jones S., *Experimental Verification of Buckling-Mode Interaction in Intermediate-Length Composite Columns*. International Journal of Solids and Structures. 2000; 37(29): 3919-3934.
- [4] Barbero E. J., *Finite Element Analysis of Composite Materials Using ANSYS*. 2nd ed. CRC Press; 2014.
- [5] Barbero E. J., *Finite Element Analysis of Composite Materials Using Abaqus*. CRC Press; 2013. pp. 91-194.
- [6] Barbero E. J., Godoy L. A., Raftoyiannis I.G., *Finite elements for three-mode interaction in buckling analysis*. International Journal for Numerical Methods in Engineering. 1996; 39(3): 469-488.
- [7] Barbero E. J., *Introduction to Composite Materials Design*. 2nd ed. CRC Press; 2010. pp. 385-393.
- [8] Barbero E. J., *Prediction of Buckling-Mode Interaction in Composite Columns*. Mechanics of Composite Materials and Structures. 2000; 7(3): 269-284.

- [9] Brush D. O., Almroth B. O., *Buckling of bars, plates, and shells*. McGraw-Hill; 1975. pp. 1-15, pp. 21-119, pp. 263-303.
- [10] Buskell N., Davies G. A. O., Stevens K. A., *Post-buckling Failure of Composite Panels*. Proceedings of the Third International Conference on Composite Structures. 1985. pp. 290-294.
- [11] Butler R., Lillico M., Hunt G. W., McDonald N. J., *Experiments on interactive buckling in optimized stiffened panels*. 1st ASMO UK/ISSMO Conference on Engineering Design Optimisation. 1999.
- [12] Cameron R., Michalk D., *Pro ASP.NET 3.5 Server Controls and AJAX Components*. APress; 2008.
- [13] Collier C., Yarrington P., *Composite, Grid-Stiffened Panel Design for Post Buckling Using Hypersizer*. American Institute of Aeronautics and Astronautics. 2011.
- [14] Cosso F. A., Barbero E. J., *Computer Aided Design Environment for Composites*. SAMPE 2012 Conference and Exhibition. 2012.
- [15] Cruz J. L., *Análisis del Comportamiento en Post-Pandeo de un Panel Curvo Rigidizado, de Material Compuesto, Mediante la Herramienta PATRAN/NASTRAN* [Senior Project]. Universidad de Sevilla; 2010.
- [16] Danielson D. A., Kihl D. P., Hodges D. H., *Tripping of thin-walled plating stiffeners in axial compression*. Thin-Walled Structures. 1990; 10(2): 121-142.
- [17] Elaldi F., *Buckling, Post-buckling and Failure Analysis of Hat Stiffened Composite Panel*. Proceedings of 8th International Fracture Conference. 2007.
- [18] Elaldi F., *Structural Efficiency and Post-buckling Strength of J- and Hat-stiffened Composite Panels*. Journal of Reinforced Plastics and Composites. 2010; 29(10).
- [19] Falzon B. G., Steven G. P., *Buckling Mode Transition in Hat-stiffened Composite Panels Loaded in Uniaxial Compression*. Composite Structures. 1997; 37(2): 253-267.
- [20] Geon-Hiu K., Jin-Ho C., Jin-Hwe K., *Manufacture and Performance Evaluation of the Composite Hat-Stiffened Panel*. Composite Structures. 2010; 92(9): 2276-2284.
- [21] Geuzaine C., Remacle J. F., *Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities*. International Journal for Numerical Methods in Engineering. 2009; 79(11): 1309-1331.
- [22] Giles T., *WebGL tutorial*. Available from: <http://learningwebgl.com/blog/?p=28>, Learning WebGL [homepage on the Internet]. 2009 [updated 2013, accessed 28 January 2014].
- [23] Godoy L. A., Barbero E. J., *Análisis y cálculo de estructuras de materiales compuestos*. CIMNE; 2002. pp. 47-83.
- [24] Godoy L. A., *Theory of Elastic Stability, Analysis and sensitivity*. Taylor and Francis; 2000.

- [25] Godoy L. A., *Thin-walled structures with structural imperfections, Analysis and Behavior*. Pergamon; 1996.
- [26] Grondin G. Y., Elwi A. E., Cheng J. J. R., *Buckling of stiffened plates - a parametric study*. Journal of Constructional Steel Research. 1999; 50(2): 151-175.
- [27] Hinton E., Owen D. R., *An Introduction to Finite Element Computations*. Pineridge Press Limited; 1979.
- [28] *Hypersizer*, Composite Analysis and Structural Sizing software [homepage on the Internet]. Collier Research Corporation; 2006 [accessed 12 March 2014]. Available from: <http://hypersizer.com/products/Concepts-Tabs.php>
- [29] Khdeir A. A., Reddy J. N., Librescu L., *Analytical Solution of a Refined Shear Deformation Theory for Rectangular Composite Plates*. International Journal of Solids and Structures. 1987; 23(10): 1447-1463.
- [30] Koiter W. T., Pignatoro M., *A general theory for interaction between local and overall buckling of stiffened panels*. Delft University of Technology. Report WTHD-83. 1976.
- [31] Le Tran K., Davaine L., Douthe C., Sab K., *Stability of curved panels under uniform axial compression*. Journal of Constructional Steel Research. 2012; 69(1): 30-38.
- [32] Li L. Y., Betess P., *Buckling of Stiffened Plates and Design of Stringers*. International Journal of Pressure Vessels and Piping. 1997; 74(3): 177-187.
- [33] Liu Y. J., Riggs H. R., *The MIN-N family of pure-displacement, triangular, Mindlin plate elements*. Structural Engineering and Mechanics. 2005; 19(3): 297-320.
- [34] Liu Y. J., *Development of the Min-N Family of Triangular Anisoparametric Mindlin Plate elements* [Ph.D. dissertation]. Manoa: University of Hawai'i; 2002.
- [35] MacDonald M., *Beginning ASP.NET 4.5 in C#*. APress; 2012. pp. 9-615.
- [36] MacDonald M., Freeman A., Szpurzta M., *Pro ASP.NET 4 in C# 2010*. APress; 2010.
- [37] Mallela U. K., Upadhyay A., *Buckling of Laminated Composite Stiffened Panels Subjected to In-Plane Shear: A Parametric Study*. Thin-Walled Structures. 2006; 44(3): 354-361.
- [38] Megson T. H. G., *Aircraft structures for engineering students*. 2nd ed. Halsted Press; 1990.
- [39] Mittelstedt C., *Explicit Analysis and Design Equations for Buckling Loads and Minimum Stringer Requirements of Orthotropic and Isotropic Plates under Compressive Load Braced by Longitudinal Stringers*. Thin-Walled Structures. 2008; 46(12): 1409-1429.
- [40] Navarro C., *Elasticidad y Resistencia de Materiales I*. Available from: http://ocw.uc3m.es/mecanica-de-medios-continuos-y-teoria-de-estructuras/elasticidad_resistencia_materialesi, OpenCourseWare-UC3M [homepage on the Internet]. Leganés: OCW-UC3M; 2008 [updated 2011, accessed 10 October 2013].

- [41] Navarro C., *Elasticidad y Resistencia de Materiales II*. Available from: http://ocw.uc3m.es/mecanica-de-medios-continuos-y-teoria-de-estructuras/elasticidad_resistencia_materialesii, OpenCourseWare-UC3M [homepage on the Internet]. Leganés: OCW-UC3M; 2008 [updated 2011, accessed 10 October 2013].
- [42] Niu M. C. Y., *Composite Airframe Structures, practical design information and data*. Conmilit Press LTD; 1992.
- [43] Niu M. C. Y., *Airframe Stress Analysis and Sizing*. Conmilit Press; 1997.
- [44] Niu M. C. Y., *Airframe Structural Design*. Conmilit Press; 1999.
- [45] Paik J. K., van der Veen S., Duran A., Collette M., *Ultimate Compressive Strength Design Methods of Aluminum Welded Stiffened Panel Structures for Aerospace, Marine and Land-Based Applications: A Benchmark Study*. Thin-Walled Structures. 2005; 43(10): 1550-1566.
- [46] Purdum J., *Beginning Object-Oriented Programming with C#*. Wrox Press; 2012. pp. 35-492.
- [47] Raftoyiannis I. G., *Buckling Mode Interaction in FRP Columns* [Ph.D. dissertation]. Morgantown (WV): West Virginia University; 1994.
- [48] Reddy J. N., *Energy Principles and Variational Methods in Applied Mechanics*. 2nd ed. John Wiley; 2002.
- [49] Reddy J. N., *Mechanics of laminated composite plates and shells, Theory and Analysis*. 2nd ed, CRC Press; 2004. pp. 109-288.
- [50] Reddy J. N., *Theory and Analysis of Elastic Plates*. Taylor & Francis; 1999.
- [51] Reddy J. N., Khdeir A. A., Librescu L., *Lévy Type Solutions for Symmetrically Laminated Rectangular Plates Using First-Order Shear Deformation Theory*. Journal of Applied Mechanics. 1987; 54(3): 740-742.
- [52] Reinoso J., *Análisis Numérico del Comportamiento a Post-Pandeo de Paneles Rigidizados de Materiales Compuestos* [Master Thesis]. Universidad de Sevilla; 2008.
- [53] Rivas E., *Herramienta computacional para diseño de paneles rigidizados de materiales compuestos* [Senior Project]. Universidad Carlos III de Madrid; 2013.
- [54] Shama M., *Buckling of Ship Structures*. Springer; 2013. pp. 45-114.
- [55] Singer J., Arbocz J., Weller T., *Experimental Methods in Buckling of Thin-walled Structures*. Buckling Experiments; 1998.
- [56] Timoshenko S. P., Gere J. M., *Theory of Elastic Stability*. McGraw-Hill; 1961.
- [57] Troitsky M. S., *Stiffened Plates: Bending, Stability and Vibrations*. Elsevier Scientific Pub. Co.; 1976.

- [58] Wang W., Guo S., Chang N., Yang W., *Optimum Buckling Design of Composite Stiffened Panels Using Ant Colony Algorithm*. Composite Structures. 2009; 92(3): 712-719.

Appendices

Appendix A: MSH file

This kind of files is used by the freeware gmsh [21]. The extension is .msh. They contain the information required by the software to define a mesh. In order to illustrate how the information has to be presented in the file, the 100x100 plate analyzed in Section 9.1.1.A will be considered. The file has the following structure:

```
$MeshFormat
2.2 0 8
$EndMeshFormat
$Nodes
81
1      50      -50      0
2      50      -37.5    0
3      50      -25      0
...
80     -50      37.5    0
81     -50      50      0
$EndNodes
$Elements
16
1      10      1      100      1      3      13      11      2      8      12      6
7
2      10      1      100      11      13      23      21      12      18      22      16
17
...
16     10      1      100      71      73      81      79      72      77      80      75
76
$EndElements
```

Note that some of the lines have been removed since they just followed the same structure as the previous lines. The file starts by defining some information required by gmsh [21] such as version number, file type or data size. We used the default values. Then, the command **\$Nodes** informs the program that the nodes of which the mesh consists are going to be presented. The immediate line after **\$Nodes** contains the total number of nodes that will be defined. Then, the following sequence is repeated for each of those nodes:

node-number x-coord y-coord z-coord

The first value is an integer going from number 1 to the total number of nodes (81 in this example). This is the ID of each node. Then, three coordinates defining the position of that node are defined based on global X, Y and Z coordinates.

After the nodes have been defined, the directive **\$Elements** appears, followed by the total number of elements. Then, a sequence of lines defining each of the elements follows. The structure of each line is the following:

elm-number elm-type number-of-tags < tag > ... node-number-list

- ***elm-number***: ID of each element, going from 1 to the total number of elements in the mesh.
- ***elm-type***: an integer number defining the type of element that is being defined. In the case of our mesh generator, the relevant numbers are: number 2 stands for first order triangular elements, number 3 for first order quadrangular elements and number 10 for second order quadrangular elements.
- ***number-of-tags***: it defines the number of tags that will be used. Tags are used in order to inform gmsh [21] whether an element belongs to a certain physical or geometrical entity. However, we are just going to use one tag so therefore our number is equal to 1. Then all the tags number would follow. In our case, just an additional number comes. The tags that will be defined in our application are the following:
 - 100: the element belongs to the skin.
 - 200: the element belongs to the bottom flange.
 - 300: the element belongs to the web.
 - 400: the element belongs to the top flange.
- ***node-number-list***: the remaining numbers account for the ID of each node defining the element. The order in which each node ID appears is very important, and this order is totally dependent on the type of element. In this example the plate was meshed using second order quadrangular elements consisting of 9 nodes. The node sequence for the three element types used by the application were previously explained in depth.

Appendix B: BMI3 INP file

The extension of this file is .inp. The file under consideration contains all the information required to solve the buckling problem with BMI3, except for the material properties (ABDH matrices), which are obtained by means of the DAT file. The structure of the file is very intuitive: first the mesh is defined, then boundary conditions and loads are specified. In order to illustrate this with an example let us consider the same 100x100 plate analyzed in Section 9.1.1.A. A summary of the content of the file follows:

```
*HEADING
SPD Program
*NODE
1,50,-50,0
2,50,-37.5,0
3,50,-25,0
...
81,-50,50,0
*ELEMENT, TYPE
1,1,3,13,11,2,8,12,6,7
2,11,13,23,21,12,18,22,16,17
...
16,71,73,81,79,72,77,80,75,76
*BOUNDARY
1,3,,0
1,5,,0
2,3,,0
2,5,,0
...
5,2,,0
45,2,,0
*CLOAD
49,2,4.166666666666667
53,2,16.666666666666667
57,2,8.333333333333333
...
41,2,-4.166666666666667
```

After the Heading tag, a description of the file comes. It does not affect the resolution of the program. Then, the mesh is defined. Nodes are described in the same way as in the msh file: the node ID is followed by its three coordinates with respect to the global CSYS. Then, elements are defined without tags: the element ID is directly followed by the nine nodes ID of which it consists. Notice that the element type is not specified since BMI3 just accepts the second order quadrangular element of 9 nodes at this moment. After the mesh description is finished, boundary conditions are specified. The structure is as follows:

Node ID, Degree Of Freedom,, value

First the ID of the node to which the BC is applied is defined, followed by the degree of freedom that will be constrained, and finally a value. Since in our cases the user will not be allowed to impose a predefined displacement, the last value will be always zero. However, it would be possible to impose displacements instead of just applying forces to the panel, but current version of CADEC is not prepared to offer this possibility to the user.

Similarly to the previous structure, loads are defined. Three values are required: node ID, DOF and finally the value of the load in that DOF.

Appendix C: BMI3 DAT file

The DAT file contains the material information of the model. The structure is the following:

```
1
1 1 1 1 1 1 1 1 1 1 1 1
1 1 1

11989.84899752981417.991905097610      0      0      0      0      0      0
1417.9919050976111989.84899752980      0      0      0      0      0      0
0      0      1822.081677474480      0      2.8421709430404E-14      0      0      0
0      0      0      330.61105335378975.62623493853890      0      0      0
0      0      0      75.6262349385389948.3061730493930      0      0      0
0      0      2.8421709430404E-14      0      0      97.17768946530570      0      0
0      0      0      0      0      0      1244.126342933020      0
0      0      0      0      0      0      0      1393.730918367050
0      0      0      0      0      0      0      0      13.9373091836705
0 0 0
```

The first line contains the number of materials present in the model. Since we are analyzing a plate, it is just one, but it could be up to four in stiffened panels. Then, a sequence of numbers follows. There are as many numbers as elements in the model. Each of those numbers represents the material type (defined with an integer from 1 to 4) that must be applied to that element. Each of the materials is applied to the following areas:

- Material 1: applied to the elements forming the skin of the stiffened panel.
- Material 2: applied to the elements forming the bottom flange of the stringers.
- Material 3: applied to the elements forming the web of the stringers.
- Material 4: applied to the elements forming the top flange of the stringers.

Of course, it is not mandatory to define all the previous areas in the file, since some of them are not present in some models (such as materials 2 to 4 in the case of a plate).

Finally, after the blank line, the ABDH matrices are defined. The last line consists of three zeros, and they inform BMI3 where the ending of the file is.

Each of the ABDH matrices has the following structure:

[A]			[B]					
A11	A12	A13	B11	B12	B13	0	0	0
A21	A22	A23	B21	B22	B23	0	0	0
A31	A32	A33	B31	B32	B33	0	0	0
B11	B12	B13	D11	D12	D13	0	0	0
B21	B22	B23	D21	D22	D23	0	0	0
B31	B32	B33	D31	D32	D33	0	[H]	0
0	0	[B]	0	0	0	H11	H12	0
0	0	0	0	0	[D]	H21	H22	0
0	0	0	0	0	0	0	0	DS

Figure C.1: combined stiffness matrices of a material (ABDH matrix).

Once the laminates have been defined, the matrices A (in-plane stiffness matrix), B (bending extension coupling matrix), D (bending stiffness matrix) and H (transverse shear stiffness matrix) are computed and their values are stored accordingly inside the ABDH array required by BMI3. It is also necessary to define the drilling factor, which does not influence much on the results but the solver needs it.

- In-plane stiffness matrix [A]: matrix (3x3) that relates directly plane stresses to membrane strains.
- Bending extension coupling matrix [B]: matrix (3x3) coupling plane stresses and bending strains. For symmetric laminates this matrix is null.
- Bending stiffness matrix [D]: matrix (3x3) that relates bending moments to bending strains.
- Transverse shear stiffness matrix [H]: matrix (2x2) connecting transverse shear with vertical loads.
- Drilling Stiffness (DS): it does not influence much on the results but the solver needs it. It is assigned a value 100 times smaller of the largest diagonal term of H matrix.

Appendix D: Abaqus INP file

This file contains all the information regarding the FE model in order to solve the buckling problem. Contrary to the BMI3 INP file, the information regarding material properties is also contained in the INP file, so that Abaqus® just requires one file. Again, the case of the 100x100 plate analyzed in Section 9.1.1.A will be used to illustrate the file's structure:

```
*Heading
**
** CADEC Stiffened Panel Design Application
**
** User's Stiffened Panel will be analyzed when subjected to Buckling conditions
** SI(mm) units (tonne, mm, s, N, MPa)
**
*Preprint, echo=NO, model=NO, history=NO, contact=NO
**
** PARTS
**
*Part, name=StiffenedPanel
*End Part
**
** ASSEMBLY
**
*Assembly, name=Assembly
**
*Instance, name=StiffenedPanel-1, part=StiffenedPanel
**
*Node
1,50,-50,0
2,50,-37.5,0
3,50,-25,0
...
81,-50,50,0
*Element, type =S9R5
1,1,3,13,11,2,8,12,6,7
2,11,13,23,21,12,18,22,16,17
3,21,23,33,31,22,28,32,26,27
...
16,71,73,81,79,72,77,80,75,76
*Elset, elset=Skin
1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16
*Elset, elset=BottomFlange
```

```

*Elset, elset=Web
*Elset, elset=TopFlange
*Nset, nset=Edge1
1,2,3,4,5,46,47,48,49
*Nset, nset=Edge2
49,53,57,61,65,69,73,77,81
*Nset, nset=Edge3
41,42,43,44,45,78,79,80,81
*Nset, nset=Edge4
1,6,11,16,21,26,31,36,41
**
*Orientation, name=Ori-Skin
1., 0., 0., 0., 1., 0.
3, 0.
*Orientation, name=Ori-BottomFlange
1., 0., 0., 0., 1., 0.
3, 0.
*Orientation, name=Ori-TopFlange
1., 0., 0., 0., 1., 0.
3, 0.
*Orientation, name=Ori-Web
1., 0., 0., 0., 0., 1.
3, 0.
** Section:Skin
*Shell General Section, elset=Skin, orientation=Ori-Skin
11989.8489975298,1417.99190509761,11989.8489975298,0,0,1822.08167747448,0,0
0,330.611053353789,0,0,0,75.6262349385389,948.306173049393,0
0,2.8421709430404E-14,0,0,97.1776894653057
*Transverse Shear
1244.12634293302, 1393.73091836705, 0
**
*End Instance
**
*End Assembly
**
** -----
**
** STEP: Step-1
**
*Step, name=Step-1, perturbation
*Buckle
5,, 5, 2000
**
**BOUNDARY CONDITIONS
**
** Name: BC-1 Type: Displacement/Rotation
**
*Boundary, op=NEW, load case=1
Assembly.StiffenedPanel-1.StiffenedPanel.1,3,,0
Assembly.StiffenedPanel-1.StiffenedPanel.1,5,,0

```



```

Assembly.StiffenedPanel-1.StiffenedPanel.2,3,,0
Assembly.StiffenedPanel-1.StiffenedPanel.2,5,,0
...
Assembly.StiffenedPanel-1.StiffenedPanel.45,2,,0
**
** LOADS
**
*CLOAD
Assembly.StiffenedPanel-1.StiffenedPanel.49,2,4.16666666666667
Assembly.StiffenedPanel-1.StiffenedPanel.53,2,16.66666666666667
Assembly.StiffenedPanel-1.StiffenedPanel.57,2,8.33333333333333
...
Assembly.StiffenedPanel-1.StiffenedPanel.41,2,-4.16666666666667
**
** OUTPUT REQUESTS
**
*Restart, write, frequency=0
**
** FIELD OUTPUT: F-Output-1
**
*Output, field, variable=PRESELECT
*End Step

```

For Abaqus®, relevant commands are preceded by an asterisk (*) instead of the dollar symbol (\$). Moreover, comments (lines that Abaqus® does not read) are preceded by two asterisks. Then, the file starts with a Heading tag, including some information about the analysis and some default configuration. Then, a part is created, and added to the assembly module, where it will be meshed. The definition of the mesh (nodes and elements) follows the exact same format used in BMI3 input file, with the only difference that it is necessary to specify the element type (\$9R5). Then some node and element sets are defined. Next, some orientations are defined. In the case of a flat plate it is not relevant, but for curved panels or stiffened panels with inclined faces such as in Omega profile it is important to define the normal direction to each element face. It is possible to rely on Abaqus® so that it computes it automatically, but it was decided to provide Abaqus® the Orientations computed by CADEC in order to avoid misunderstandings. Last information that must be provided inside the assembly module is the material properties definition. The values correspond to the upper diagonal terms of the ABD matrix, as follows:

1 ↓	2 ↓	3 ↓	4 ↓	5 ↓	6 ↓
A11	A12	A13	B11	B12	B13
A21	A22	A23	B21	B22	B23
A31	A32	A33	B31	B32	B33
B11	B12	B13	D11	D12	D13
B21	B22	B23	D21	D22	D23
B31	B32	B33	D31	D32	D33

Figure D.1: Terms to be introduced in Abaqus® to define laminate material properties.

The numbers next to the orange arrows determine the order in which values must be introduced (that is, values from top to bottom and columns from left to right). Therefore, the numbers contained in the file follow the sequence $A_{11}, A_{12}, A_{22}, A_{13}, \dots, D_{33}$.

After the geometry, mesh and material properties are defined, it is time to move to the analysis type. In the step, we define that a Buckling analysis will be performed, asking for the first 5 modes, and allowing a maximum number of iterations of 2000.

Next, boundary conditions and loads are defined using the same format as in the BMI3 input file. Finally some more information regarding the output is provided, and then Abaqus® has all the required information to start solving the eigenvalues problem.

Appendix E: Stored procedures and SQL language

In order for CADEC to communicate with the database, it is necessary to use SQL language (Structured Query Language). By means of a few instructions in SQL it is possible to insert new records inside a table, to read the table or to delete information. It is possible to implement this code directly inside the code-behind (along with C# code) of the application. Since this would lead to more code and less centralization, the SQL code necessary to manage the DB will be contained in files called stored procedures, and they will be stored inside the database itself. A stored procedure is a subroutine available to applications that access a relational database system. They consolidate and centralize logic. As it was previously explained, six stored procedures will be defined for managing Stiffened Panels database. The extension of these files is .sql. In order to illustrate the structure of these files, the stored procedure in charge of creating a new panel in the database will be analyzed.

First of all, it is necessary to specify the database that will be updated or modified, which is called *CADEC_DB* in this case. Then, the following lines set some default values before an *IF* statement comes. The purpose of this conditional is to create a new procedure in case it does not exist, or to update the existing procedure. This portion of code is always placed before the actual definition of the procedure. The main purpose is to allow an easy update of a procedure. Imagine that we decide to add a new property to the stiffened panel. Then, this procedure must be updated as well in order to store the new property in the DB as well once we create a new stiffened panel. With the previous code, it is just necessary to make the corresponding changes in the definition of the procedure and then run this SQL code so that the procedure is updated. If we were to add this stored procedure to a new database (with no stored procedures regarding stiffened panels), the same code would be able to create a new procedure instead of updating a new one.

Then the actual definition of the procedure starts. After the *ALTER PROCEDURE* statement, the different properties of the panel are defined, specifying the variable type of each record. Then, since a new stiffened panel is being added to the table in the DB, a unique ID is necessary to represent that panel. This can be achieved with the instruction *SELECT @Id = NEWID()*. Finally, the value of each property (values coming from the page) is assigned to the corresponding column (each record in the DB).

This was an example about how to add a new row in the database (new panel). The structure of this file is very similar to the one of the remaining procedures. The main differences are regarding the instruction used:

- Creation of a new record (INSERT instruction).
- Deletion of an existent record (DELETE instruction).
- Edition of some fields of a record (UPDATE and SET instructions).

- Retrieval of a single field of a record from the DB (SELECT instruction).
- Retrieval of a list of all the records (SELECT and WHERE instructions).
- Retrieval of a complete record of the DB, including all the different fields for that record (SELECT and WHERE instructions).